P4CONTROL: Line-Rate Cross-Host Attack Prevention via In-Network Information Flow Control Enabled by Programmable Switches and eBPF

Osama Bajaber
Virginia Tech
obajaber@vt.edu

Bo Ji
Virginia Tech
boji@vt.edu

Peng Gao
Virginia Tech
penggao@vt.edu

***Abstract*—Modern targeted attacks such as Advanced Persistent Threats use multiple hosts as stepping stones and move laterally across them to gain deeper access to the network. However, existing defenses lack end-to-end information flow visibility across hosts and cannot block cross-host attack traffic in real time. In this paper, we propose P4CONTROL, a network defense system that precisely confines end-to-end information flows in a network and prevents cross-host attacks at line rate. P4CONTROL introduces a novel *in-network decentralized information flow control (DIFC) mechanism* and is the first work that enforces DIFC at the network level at *network line rate*. This is achieved through: (1) an in-network primitive based on programmable switches for tracking inter-host information flows and enforcing line-rate DIFC policies; (2) a lightweight eBPF-based primitive deployed on hosts for tracking intra-host information flows. P4CONTROL also provides an expressive policy framework for specifying DIFC policies against different attack scenarios. We conduct extensive evaluations to show that P4CONTROL can effectively prevent cross-host attacks in real time, while maintaining line-rate network performance and imposing minimal overhead on the network and host machines. It is also noteworthy that P4CONTROL can facilitate the realization of a zero trust architecture through its fine-grained least-privilege network access control.**

1. Introduction

Despite the dramatic growth in expenses on operational network security, we are still witnessing a rapid increase in targeted cyber attacks, such as Advanced Persistent Threats (APTs). These sophisticated attacks often exploit multiple hosts in a network and laterally move to the target to access unauthorized resources or exfiltrate sensitive data [1]. As a result, many high-profile businesses were plagued with huge losses [2]. These *cross-host attacks* pose significant challenges to existing defenses, which lack the necessary context to correlate attack activities on different hosts and prevent attacks from damaging the network in real time.

Existing defenses treat inter-host information flows and intra-host information flows in isolation. Hence, they lack ***end-to-end information flow visibility*** across multiple hosts in a network. Network-level defenses, such as firewalls [3] and network intrusion detection systems (NIDSes) [4], have visibility into inter-host information flows between two hosts in the form of network flows (i.e., a sequence of packets sent from a source to a destination [5]). However, they are unable to connect network flows to reveal cross-host attack activities due to lack of host-level visibility on intermediate hosts. On the other hand, host-level defenses capture intra-host information flows only. Many studies along this line employ system call monitoring to track information flows between system entities (e.g., processes, files) within a host for forensic investigation [6]–[10]. However, they are unable to track attack activities beyond a single host due to inadequate network-level visibility. Although a few studies along this line proposed to associate system calls across hosts [11]–[14], these solutions mostly operate in post-compromise settings using historical system audit logs. In summary, none of the existing defenses are able to block cross-host attack traffic in real time *when the connection is established on the fly*.

Consider a representative enterprise network scenario (see Fig. 1a). Attackers can bypass the firewall to access sensitive data on the protected host, `Server1`, by laterally moving across four hosts with multiple inter-host information flows and intra-host information flows.

Goal and challenges. To that end, the overarching goal of this work is to design and build a new network defense system that (1) enables end-to-end information flow visibility across hosts, and (2) leverages such visibility to enforce security decisions in real time to prevent cross-host attacks. The key challenge to achieving this goal is three-fold:

First, to enable end-to-end visibility, an effective defense must accurately correlate information flows both within and between hosts in a cross-host attack. Furthermore, the defense must also *precisely confine* the information flow among entities (e.g., hosts, packets, processes, and files) and enforce authorized accesses. While decentralized information flow control (DIFC) [15] provides the needed fine-grained control of information flow, existing DIFC systems have only focused on operating systems (OSes) [16]–[19], distributed systems [19], [20], and cloud computing [21], [22]. None of them have enforced DIFC *at the network level*.

Second, enforcing DIFC at the network level is highly challenging due to the huge volume of traffic in enterprise networks. An effective defense must be able to correlate

information flows and enforce DIFC policies on the fly, without imposing significant overhead on the network performance. The defense must be seamlessly integrated with the existing network infrastructure and must not affect the *line-rate processing* of large amounts of benign traffic.

Third, human analysts with domain knowledge are crucial for defenses [10]. An effective defense must hide the complexity of low-level DIFC enforcement and allow the network administrator to tailor the defenses for different attacks, through a flexible and expressive policy interface.

Contributions. We propose P4CONTROL, a network defense system that precisely confines end-to-end information flows and prevents cross-host attacks in real time when the connection is established on the fly. P4CONTROL introduces a novel ***in-network DIFC mechanism*** for precise information confinement and line-rate DIFC policy enforcement. The mechanism includes a secure network-level DIFC model with a category system and an in-network DIFC enforcement approach enabled by programmable switches and eBPF. P4CONTROL also provides a flexible and expressive policy framework to specify a wide range of DIFC policies.

P4CONTROL creates DIFC labels for network entities (e.g., hosts and packets) and system entities (e.g., processes and files) and propagates these labels along intra-host and inter-host flows between the entities. These labels can encode different *categories, secrecy and integrity levels, etc.* P4CONTROL then enforces DIFC policies specified by the network administrator on labeled flows in the *network data plane* at line rate. In addition, P4CONTROL provides (1) safe mechanisms to declassify secret data to authorized readers or endorse data as high integrity, and (2) a tainting mechanism for fine-grained tracking of the propagation path of a sensitive file in the network to limit its reachability. Our network-level DIFC model formalizes these operations.

P4CONTROL uniquely leverages the emerging programmable switches and eBPF to realize the DIFC model in the data plane. Programmable switches offer data-plane programmability through P4 [23] and guarantee customized Tbps line-rate packet processing. This enables P4CONTROL to process labeled network traffic and enforce line-rate DIFC policies. To address the key challenge of limited switch memory and minimize the network overhead, P4CONTROL employs a secure-yet-practical in-network DIFC enforcement approach with *tailored techniques* to label network traffic, match DIFC policies, enforce per-flow decisions, and enable declassification/endorsement controls.

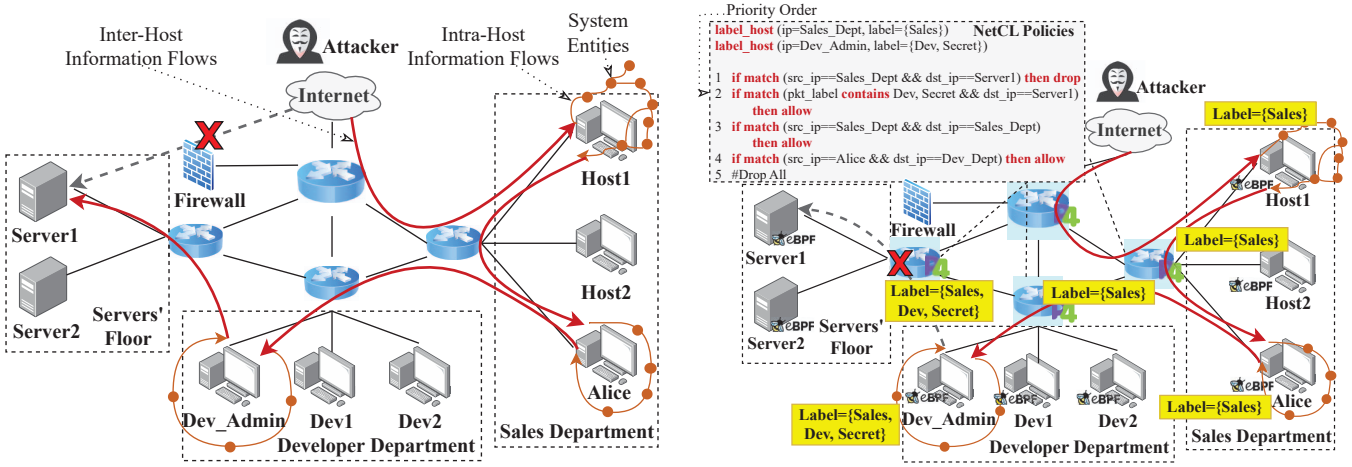
To propagate DIFC labels within each host and from/to the network, we develop a lightweight host agent based on the eBPF technology [24]. Our host agent is lightweight and readily deployable *without any kernel modifications*. This differentiates our approach from previous DIFC works [17], [18], [25], which require extensive OS kernel modifications to track intra-host information flows. Our host agent enables *lightweight DIFC label persistence* by attaching carefully defined eBPF hooks in the kernel to capture the complete chain of intra-host system events and accurately propagate DIFC labels, with minimal overhead on the host machine.

While programmable switches enable in-network DIFC, it is challenging for network administrators to directly program the data plane using P4, which is low-level and can be error-prone [26]. To unlock the powerful in-network DIFC context, we design *Network Control Language (*NETCL*)*, an expressive domain-specific language that enables the network administrator to specify DIFC policies that match cross-host flows and trigger a wide range of defense actions, including preventing data exfiltration, detecting unauthorized access, declassifying information, and limiting the reachability of sensitive files and the spread of malware. NETCL policies follow a priority-based enforcement similar to the traditional firewall policies. To further enhance the defense agility against attacker's possibly changing strategies, P4CONTROL employs an efficient compilation mechanism that supports *dynamical update* of NETCL policies at runtime without interrupting network traffic.

Evaluation. We extensively evaluated P4CONTROL's defense effectiveness and coverage, scalability, system capacity, and system overhead, and compared with firewall (iptables [3]), NIDS (Snort [4]), and SDN-based IFC solutions (PivotWall [25]). We deploy P4CONTROL on both a physical testbed (with a Tofino programmable switch [27]) and representative large enterprise topologies constructed using a packet-level simulation, and use synthetic and real-world enterprise traces. The evaluation results demonstrate that: (1) P4CONTROL successfully prevents a wide range of stealthy cross-host attacks that all evade firewall and NIDS, while adding only a negligible $\sim$110 ns overhead to the packet processing time. (2) P4CONTROL successfully prevents real-world enterprise cross-host attacks from the DARPA OpTC dataset [28] and the LANL Unified Host and Network dataset [29], while achieving 99.9 Gbps throughput on the 100 Gbps (per-port) programmable switch, incurring minimal overhead on benign traffic. (3) P4CONTROL is robust against control plane attacks and achieves robust performance even with an attack strength of 1 million packets/second, compared to PivotWall, which fails to install 99% of legitimate connections. (4) The eBPF-based agent adds an overhead of 1-7 ms to the total time of the monitored system calls, imposing minimal overhead on the host performance.

Clearly, these results demonstrate that P4CONTROL outperforms existing defenses in combating cross-host attacks by a significant margin. P4CONTROL's in-network DIFC approach based on programmable switches and eBPF offers robust defensive effectiveness and wide attack coverage, maintains line-rate performance, and imposes minimal overhead on the network and host machines. We open-source the prototype of P4CONTROL at [30].

Significance of the work. P4CONTROL is the first work that enforces DIFC at the network level at line rate. P4CONTROL is also the first work that uses programmable data planes to enforce complex secrecy and integrity policies at line rate. P4CONTROL introduces a new paradigm of network-level APT defenses using programmable data planes, which differs from all existing system-level APT defenses based on system audit logs and system provenance graphs (e.g., [6]–



(a) Motivating cross-host attack example

(b) Defense workflow of P4CONTROL

Fig. 1: (a) A real-world attack scenario: while the firewall can block any direct connections (indicated by the grey dashed arrow) from the external network to a protected server, `Server1`, by exploiting intermediate hosts, the attacker can successfully bypass the firewall and reach its final target with four inter-host information flows (indicated by the red arrows) and multiple intra-host information flows (indicated by the orange arrows). (b) An illustration of the defense workflow of P4CONTROL deployed in a network of programmable switches: P4CONTROL parses DIFC labels (indicated in the yellow boxes) to precisely correlate and confine information flows across hosts and blocks cross-host attack traffic in real time based on the priority-ordered DIFC policies.

[14]). P4CONTROL can be seamlessly integrated into the existing network infrastructure with minimal modifications and overhead, transforming it into a defense backbone. P4CONTROL radically shifts from the traditional "castle-and-moat" security model that relies on perimeter defenses and implicit trust inside the network, and aligns with the principles of zero trust. By precisely confining information in a network with DIFC labels, P4CONTROL can facilitate the realization of a *zero trust architecture* [31] through its fine-grained least-privilege network access control.

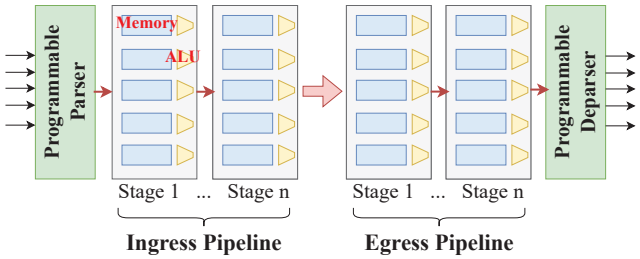


Fig. 2: Protocol independent switch architecture

2. Background and Motivating Example

Motivating example. Organizations face challenges in defending their resources against cross-host attacks like APTs, particularly as networks become larger and more complex [1]. Consider an enterprise network (see Fig. 1a). Hosts and servers across different departments are interconnected, each protected by a perimeter firewall. Alice, a former developer, transitions to a new role in the Sales Department, while retaining access to the Developer Department to complete a project. Users like Alice (i.e., those with multi-domain access) are common in enterprise networks.

An external attacker aims to compromise the integrity of `Server1` and exfiltrate data. To protect `Server1`, the firewall is configured to only allow direct connections between `Dev_Admin` and `Server1`. If the attacker attempts to directly connect to `Server1` from the external network, the attempt will be blocked (i.e., grey dashed arrow). However, the attacker can target `Alice`'s dual access as a *stepping stone*: (1) infiltrating `Host1` by exploiting a zero-day vulnerability through spear-phishing; (2) pivoting connections and gaining access to `Alice`, `Dev_Admin`, and `Server1` in sequence; (3) launching a ransomware attack (compromising integrity) or data exfiltration (compromising confidentiality) on `Server1`. The attack path consists of four inter-host flows (i.e., red arrows) and multiple intra-host flows (i.e., orange arrows). Since firewalls can only observe coarse-grained information (e.g., IP addresses and port numbers) and use such limited information to block *direct* communications between two hosts, they miss such stepping-stone attacks.

Programmable data plane as a defense solution. Programmable switches have recently attracted increased attention for their data-plane programmability that achieves line-rate performance with low overhead, compared to the software-defined networking (SDN) counterparts that often require extensive control-plane communications. These switches can be programmed using P4 [23] for customizing packet processing (thus also called P4 switches). As long as a P4 program can be successfully compiled, the data plane guarantees to process packets at *Tbps line rate* [27].

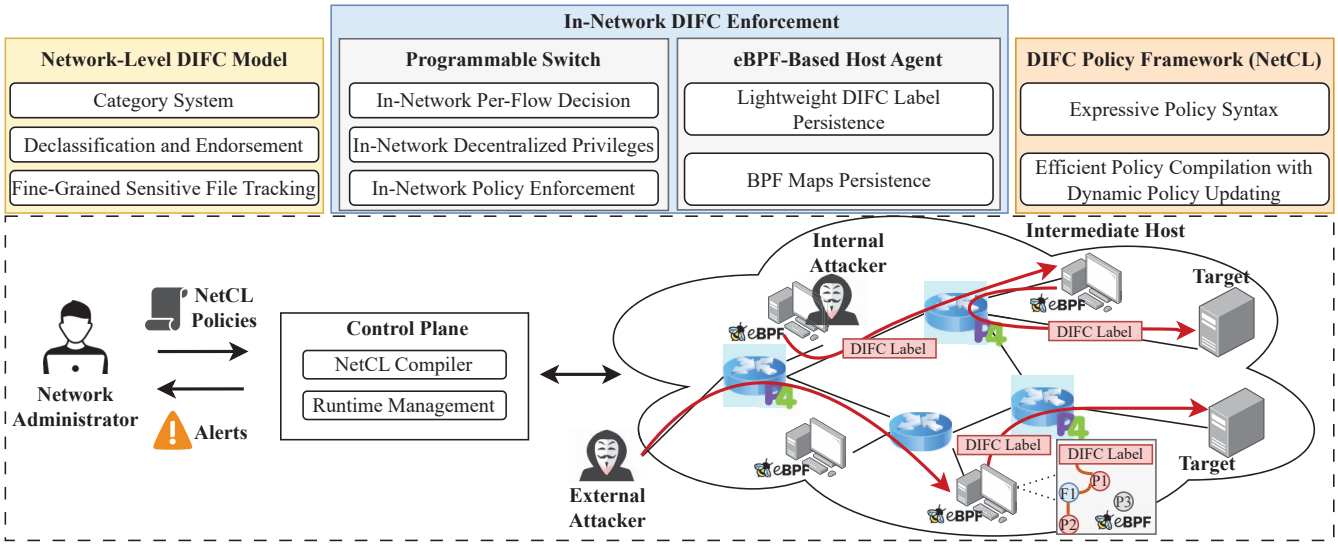


Fig. 3: Overall architecture of P4CONTROL

Fig. 2 illustrates the switch architecture. P4 programs specify the packet headers and the operations on these headers. The programmable parser parses the user-defined packet headers. These headers go through multiple hardware stages with Arithmetic Logic Units (ALUs) and match/action tables, where match fields and types (e.g., exact/range/ternary matching) can be specified. These stages use Static RAM (SRAM) and Ternary CAM (TCAM) for match lookups. While SRAM supports exact matching and persists data across packets for stateful processing, TCAM supports wildcard matches over header fields. Unfortunately, *switching ASICs only offer limited memory* (e.g., hundreds of MB of SRAM and tens of MB of TCAM) [27]. To guarantee line-rate processing, P4 programs limit the operations in each stage. When the packet header matches a table, it triggers an action. The programmable deparser reassembles the packet before it leaves the switch and gets forwarded.

Recent works proposed different implementations inside the programmable switch to defend against distributed denial-of-service (DDoS) [32], [33], link flooding [34], [35], covert channels [36], and BYOD security [37]. However, none of these works have focused on cross-host attacks.

DIFC. Information flow control (IFC) monitors and regulates the movement of information within a system. In classical centralized IFC [38]–[40], a central authority assigns predefined security labels to subjects (e.g., processes) and objects (e.g., files) and enforces IFC policies. Only the central authority can change labels or policies. Decentralized IFC (DIFC) [15] is a generalization of the classical IFC and offers more flexibility and autonomy. In DIFC, data owners can set their own security policies and labels for their data, and grant permissions to subjects to alter their labels.

While prior works have integrated DIFC with OS [16]–[18] and extended it to Android [41] and distributed systems [19], [20], these implementations require substantial modifications to the kernel or user-space application, which is highly complex. Besides, their enforcement happens within individual hosts rather than in the network. These DIFC systems also incur a significant overhead on system operations (e.g., 30-40% slower in the Flume system [16]) due to the low processing power of host systems, failing to meet our line-rate requirement. While other works have employed IFC in cloud environments [21], [22], these works focus on the interactions between users and cloud providers, which are distinct from network communications.

3. P4CONTROL System Overview

We present the overall architecture of P4CONTROL in Fig. 3 and briefly describe its defense workflow below.

Initialization: (1) The network administrator specifies NETCL policies for assigning DIFC labels (i.e., a set of DIFC tags) to hosts and matching network flows. (2) The programmable switch sends a control packet containing the specified DIFC label to the corresponding eBPF-based host agent. (3) The host agent initializes existing processes and files in the host with the received DIFC label.

DIFC context persistence: (4) As an attacker enters the host system, the host agent propagates the DIFC label between system entities along with the attacker's activities. When the attacker pivots to another host, the host agent propagates the DIFC label to the network by incorporating the label into the outgoing network flow (in a customized DIFC packet header). (5) The host agent on the receiver host extracts the DIFC label from the network flow, merges it with the label of the receiver process, and continues propagating the updated label. This way, the host agents maintain the DIFC context persistence across the network.

DIFC policy enforcement: (6) NETCL policies specified by the network administrator are compiled into different in-network policies, which are inserted into the programmable switch's match/action tables. (7) When a labeled network

flow arrives at the switch, the switch extracts DIFC tags from the DIFC label and uses these tags to correlate all previous flows. The switch then matches these tags against the in-network policies to trigger the corresponding security action (e.g., drop the flow) at line rate.

Fig. 1b illustrates how P4CONTROL counters the attack scenario described in Section 2. After initialization, the processes on `Dev_Admin` have the label {`Dev`, `Secret`} with the tags `Dev` and `Secret`. When the attacker pivots from the Sales Department to `Dev_Admin`, the {`Sales`} label is propagated to the network flow, and the attacker process on `Dev_Admin` then has the label {`Sales`, `Dev`, `Secret`}, indicating that the process has previous interactions with entities that have the `Sales` tag. Subsequently, when the attacker tries to connect to `Server1`, the programmable switch detects the presence of the `Sales` tag in the network flow and realizes that the flow traverses the Sales Department. The switch then enforces the matched policy that has the highest priority (i.e., the `drop` policy) to drop the traffic.

Threat model. Our threat model is similar to that of many previous works on programmable switch-based network defenses [32]–[37] and host-level auditing [6]–[14]. We assume the presence of an attacker seeking to access or modify unauthorized resources, exfiltrate confidential data, or spread malware, either from within the network or externally, by exploiting trust relationships among networked hosts. Our trusted computing base includes programmable switches, the control plane, and host agents. We assume that the OS kernels are secure from compromise, and that the network administrator specifies policies correctly especially regarding declassification. We do not consider malicious administrators who can disable the host agent or tamper with DIFC labels, or implicit flows like covert and timing channels. Tamper-proof and tamper-evident auditing techniques [42], [43] can be leveraged to further secure our host agents.

4. Network-Level DIFC Model

P4CONTROL implements a secure network-level DIFC model with a category system to associate entities with DIFC labels, a declassification and endorsement mechanism, and a mechanism for fine-grained tracking of sensitive files.

4.1. Category System

Our category system associates different network entities (e.g., hosts and packets) and system entities (e.g., processes and files) with DIFC tags and DIFC labels. We extend the Flume DIFC model [16], a host-level DIFC model, to the network level and *inherit* its security guarantees. As in Flume, P4CONTROL uses DIFC tags to govern the flow of information between processes and files residing on the same machine and processes residing on different machines. Tags are assigned to both subjects (processes) and objects (files). Directories are treated as files. A set of tags form a DIFC label [40]. These tags and labels can encode various categories (e.g., different enterprise departments) and secrecy and integrity levels (e.g., top-secret, secret, and unclassified) for entities to achieve enhanced multi-level security, adhering to the principle of least privilege [44].

Let S_p and I_p be the secrecy and integrity labels of entity p, respectively, and let $L_p = S_p \cup I_p$ be its overall label. Following Flume's *safe message rules*, process p can send a message to process q only if $S_p \subseteq S_q$ (i.e., "no read up, no write down" [38]) and $I_p \supseteq I_q$ (i.e., "no read down, no write up" [39]). To extend the label visibility from a single host to the network, P4CONTROL incorporates the *labeling of network packets*. When a message m is sent from process p to process q, a label L_m is assigned to m. For the message m to be delivered to q, it must satisfy the condition $L_p \subseteq L_m \subseteq L_q$ before delivery.

To comply with the safe message rules, processes must change their labels before they can communicate with other processes or files. For example, a process that carries a `Sales` tag can only share data with processes having a matching `Sales` tag. Note that in Flume [16], explicit label change requires the prediction of communication patterns of subject processes to adjust labels. However, this approach is impractical in unpredictable environments and requires significant effort to modify all applications' code, limiting the DIFC's efficacy. Therefore, P4CONTROL adopts implicit label change as in Asbestos [18], allowing *implicit label propagation* between processes and files: If process p communicates with process q, then both of their labels merge to update L_q (i.e., $L_q = L_p \cup L_q$). If processes p and q are on different machines, p appends its label to the outgoing packets, which is then propagated to process q upon arrival. For files, if a process p reads from an existing file f, it initiates a flow from f to p, propagating f's label to p. This confirms that p has accessed data tagged with L_f. When p writes to a new file f, p specifies L_f for f, which includes all tags in L_p. This design is vital in tracking long-going attacks that involve data theft stored for future exfiltration.

4.2. Declassification and Endorsement

Our model supports decentralized privileges to declassify (remove secrecy tags) or endorse (add integrity tags) information. Each tag t has two associated capabilities: t^+ allows a process to add tag t to its label, and t^- allows removing tag t. Let C_p be the set of capabilities that process p has. Process p can add (or remove) tag t to its label only if it has the capability $t^+ \in C_p$ (or $t^- \in C_p$). For secrecy, the capability t^- allows a process to declassify information associated with tag t. For integrity, the capability t^+ allows a process to endorse its state with an integrity level associated with the tag t. As remote hosts are untrusted, they are modeled as an untrusted process x with an empty label (i.e., $L_x = \{\}$). Therefore, to interact with the outside world, a process must have the capability to reduce its label to $\{\}$.

4.3. Fine-Grained Tracking of Sensitive Files

The model we have described does not provide enough granularity to track individual high-value files. When a sen-

sitive file is declassified, it is hard to regulate its accessibility to unauthorized readers. To address this, we further enhance our DIFC model with a special `TrackerID` tag and a *tainting mechanism* for specific files. If a process reads a tagged file, it inherits the `TrackerID`, which is then propagated to other processes and to the network when the file data is exported. This alerts the programmable switch that a file with the `TrackerID` tag is being transmitted. This design offers two significant benefits. First, `TrackerID` enables fine-grained tracking and policy enforcement on specific sensitive files. Second, we can monitor `TrackerID` to create a provenance graph, which is useful for tracking declassified sensitive files and forensic analysis.

5. In-Network DIFC Enforcement

P4CONTROL realizes the network-level DIFC model in the data plane: P4CONTROL leverages eBPF to realize the implicit label propagation within each host and between hosts to maintain the DIFC context persistence. P4CONTROL leverages programmable switches to further *regulate* the label propagation between hosts by parsing the DIFC label carried in the network flow and enforcing line-rate DIFC policies. To minimize the network overhead, P4CONTROL employs an in-network per-flow decision mechanism that enforces DIFC policies at the flow granularity, removing the need for labeling and matching every packet in the flow. P4CONTROL also employs a multi-table flow matching technique to support a large number of in-network policies with limited switch memory.

5.1. In-Network Per-Flow Decision

To carry the DIFC label in network traffic, P4CONTROL employs a customized network packet format (see Fig. 4). We set the reserved bit in the IP fragment field (known as the "evil" bit) to distinguish labeled packets from regular packets and use a *DIFC packet header* to carry DIFC tags. These tags can be extracted by the programmable switch. Our implementation considers a 32-byte DIFC packet header, which supports 256 distinct tags (each bit represents a tag). This indicates the number of categories and security levels in the network that can be supported by the Tofino 1 switch model (Tofino 1 model supports a maximum of 256-bit matching keys within TCAM). It is noteworthy that this capacity largely exceeds the U.S. Department of Defense minimum access control requirement of 16 sensitivity classifications and 64 categories [45]. The latest switch models (e.g., Tofino 2/3 [46], [47]) have 3× more resources than Tofino 1 and can support a larger number of tags.

Per-flow decision. A naive way of carrying DIFC labels is to label every packet in a network flow. However, this would waste resources, as the same security decision applies to all packets in the same network flow. To reduce the network overhead, P4CONTROL employs an *in-network per-flow decision* mechanism using stateful registers in programmable switches. Rather than labeling every packet in a network flow, P4CONTROL only adds the DIFC packet header to the *initial packets* of a flow. The security decision for the flow is then maintained in a match/action table (called `ConnDec` table), which includes the flow's 5-tuple key ($\mathrm{IP_{src}}$, $\mathrm{Port_{src}}$, $\mathrm{IP_{dst}}$, $\mathrm{Port_{dst}}$, Protocol) and the decision value. Subsequent packets in the flow will match the corresponding entry in the `ConnDec` table, and the same decision will be applied.

Fig. 4: Headers of DIFC-labeled network packet

P4CONTROL employs different strategies to *support different network protocols*. For TCP connections, the host agent adds the DIFC packet header to the SYN packet during the three-way handshake. This guarantees that the label is received by the switch for successful connections. However, in UDP connections, where packet delivery is not guaranteed, the switch may not receive the packet that carries the label. To address this, the host agent adds the DIFC packet header to the first few UDP packets in a new connection. Once the switch receives a packet with the DIFC packet header, it crafts an ACK packet using the hardware packet generator and sends it back to the sender host. This acknowledges that the DIFC label has been received, allowing the host to send the remaining packets without additional DIFC packet headers. For ICMP, the host agent adds the DIFC packet header to the request and reply ICMP packets.

Hardware decision buffering. An issue arises when relying solely on the `ConnDec` table. Although match/action tables can handle a large number of entries, the control plane must be involved to add entries to `ConnDec` for every new network flow. This means that the switch has to request the control plane to install an entry after matching a new flow, introducing a delay, known as round-trip time (RTT), from when the switch matches a decision for a new flow to the point where the entry is inserted into `ConnDec`. During this time period, the remaining packets of the matched flow can arrive at the switch before their entry is inserted.

To address this issue, we implement a *hardware buffer* structure using stateful registers, which can be directly updated by the switch's data plane at line rate. When a new network flow arrives, P4CONTROL matches the flow using the DIFC label in the flow's first packet, inserts an entry to the buffer on the fly, and sends a request to the control plane to update `ConnDec`. Each entry in the buffer stores the CRC hash value of the flow's 5-tuple key and the security decision. When the remaining data packets arrive at the switch, P4CONTROL calculates their hash values and matches them with the buffered decision until the corresponding entry is inserted into `ConnDec`.

Note that hash collisions can happen. If a new network flow, `flow2`, has a collision with an existing flow, `flow1`, in the buffer, P4CONTROL evicts `flow1`'s entry to make room for `flow2`. However, `flow1`'s entry might be evicted before its corresponding entry is inserted into `ConnDec`. This can happen if `flow2` and `flow1` arrive at the switch within a very short time (i.e., RTT) and have the same hash key `h`. To

address this issue, P4CONTROL recirculates the remaining packets of `flow1` for a time exceeding the expected RTT to ensure that `flow1`'s entry is inserted into `ConnDec`.

However, it is worth noting that such recirculation rarely happens as `flow2` and `flow1` need to (1) have a collision, and (2) arrive at the switch within RTT (typically in milliseconds). Otherwise, `flow1`'s entry is already inserted into `ConnDec`, and thus, the remaining packets of `flow1` can bypass the buffer checking, and its entry in the buffer can be safely evicted. Our current implementation uses a buffer with up to 2^{32} entries, utilizing the output of CRC-32 hash function as the key. The latest Tofino 2 hardware [46] has more resources and can accommodate a buffer with up to 2^{64} entries, further reducing the chance of collisions.

Mitigating flooding attacks. Although P4CONTROL's in-network defense effectively shields against cross-host attacks, it remains critical to mitigate the potential risk of exploitation posed by malicious hosts. For example, the attacker can exhaust the stateful storage of the `ConnDec` table by initiating many new connections. To counter this, P4CONTROL employs a *rate-limiting* strategy that restricts the number of requests from an IP address over a certain period. P4CONTROL also periodically removes inactive connections from `ConnDec` to avoid resource exhaustion.

5.2. In-Network Decentralized Privileges

We now describe how P4CONTROL realizes the decentralized privileges capabilities. Note that P4CONTROL does not focus on regulating communications within hosts, which has been extensively studied in existing OS-level DIFC works [16]–[18]. Hence, for intra-host communications, P4CONTROL uses eBPF programs to implicitly propagate labels by adding tags to the relevant BPF maps to satisfy the safe message rules. This allows the information to flow freely within a host according to the subject's choice.

For inter-host communications, P4CONTROL supports information declassification (or endorsement) controls by removing (or adding) tags in packets. In existing OS-level DIFC systems, processes on hosts are responsible for declassifying or endorsing tags [16]. However, this approach can potentially overwhelm the hosts, especially for high-traffic networks, where the CPU can become a performance bottleneck when handling high-volume requests. Therefore, P4CONTROL offloads the tag capabilities to the high-speed programmable switches according to the defined NETCL policies. For a network flow that passes through the switch, the switch modifies the labeled packet header on the fly by removing secrecy tags (for declassification) or adding integrity tags (for endorsement).

The switch often needs to modify multiple tags at once. For example, when a process exports data to the external world, the label for the network flow, potentially containing several tags, needs to be downgraded to an empty label $\{\}$. To efficiently realize this, P4CONTROL uses *bitmasks*. When an in-network policy is hit, the switch receives a bitmask `mask` indicating which tags to modify (bit set to 1). For declassification, the switch performs a bitwise AND operation between the DIFC packet header and $\sim$`mask` to clear secrecy tags. For endorsement, the switch performs a bitwise OR operation with `mask` to add integrity tags.

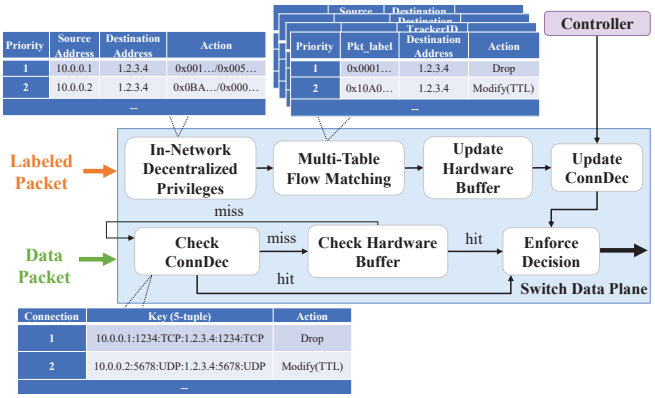


Fig. 5: In-network packet processing workflow

5.3. In-Network Policy Enforcement

NETCL policies defined by the network administrator are compiled into different *in-network policies* to be executed in the switch. For example, one type of in-network policy performs DIFC label pattern matching, which matches the network flow by examining the specific DIFC tags contained in the flow's DIFC packet header. Other types of policies include matching by the flow's source and destination hosts and matching by the `TrackerID` tag. To store these in-network policies in a switch, a naive way is to use a single large match/action table, similar to the traditional firewall structure. However, this design is highly inefficient as the policies that perform DIFC label pattern matching require ternary matching, which is expensive for DIFC packet headers of 32 bytes [48] and must be placed in TCAM. Since TCAM has a much smaller capacity than SRAM, placing all policies in a single table in TCAM would quickly exhaust its capacity, resulting in only a few hundred policies that can be stored.

To store these policies efficiently within limited switch memory, P4CONTROL employs a *multi-table flow matching* technique. Our idea is to place in-network policies in multiple match/action tables in different types of memories based on the type of matching. While policies that need DIFC label pattern matching are placed in TCAM, the other policy types that need exact matching are placed in the match/action tables in SRAM. Our evaluation result in Section 7.3 shows that this approach increases the policy storage capacity by 12$\times$ compared to the single-table design.

Priority-based enforcement. Inspired by the firewall policy design, the network administrator can define the priority of NETCL policies, and P4CONTROL maintains the priority of each policy in its table entry. For example, in Fig. 1b, the `drop` policy (order 1) has a higher priority than the `allow` policy (order 2). If a network flow matches multiple entries across different match/action tables, the switch will

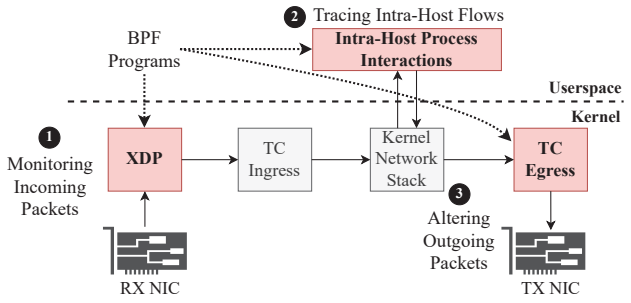


Fig. 6: eBPF hooks and intra-host label propagation path

execute the one with the highest priority. If none of the policies match, the network flow will be dropped by default, resembling the default deny action of firewalls.

In-network packet processing workflow. As illustrated in Fig. 5, labeled packets (which carry a DIFC packet header) undergo initial matching against declassification/endorsement policies for tag modifications, followed by matching against in-network policies for a security decision. The decision is then stored in the hardware buffer and the `ConnDec` table in the data plane. Subsequent packets in the network flow (which do not carry a DIFC packet header) are processed against the pre-determined decision stored in the data plane, ensuring quick handling of subsequent packets.

5.4. Lightweight eBPF-Based Host Agent

Our host agent persists DIFC context both within the host and from/to the network with minimal overhead. Enabled by eBPF (Extended Berkeley Packet Filter) [24], our host agent is lightweight and readily deployable without any kernel modifications. This differentiates our approach from previous DIFC works [16]–[18], which require extensive kernel modifications to track intra-host information flows. eBPF is an emerging kernel technology that enables sandboxed programs to run in the kernel space *without* modifying the kernel source code or loading additional modules. It enhances the performance, security, and flexibility of the kernel by allowing dynamic and event-driven programming.

Lightweight DIFC label persistence. Our host agent maintains lightweight label persistence by attaching carefully defined eBPF hooks in the kernel to capture the complete chain of intra-host events and accurately propagate DIFC labels. This design makes our host agent compatible with a wide range of kernel versions, facilitating deployment in large, heterogeneous networks with various system configurations. Fig. 6 illustrates these eBPF hook points. We create multiple BPF maps to share data between these eBPF programs.

❶ *Monitoring incoming packets:* The first hook monitors incoming labeled packets and extracts DIFC labels. To achieve high-performance packet processing, we leverage the XDP (eXpress Data Path) [49] technology to directly attach the eBPF program to the network device. When a new packet arrives, a callback invokes the eBPF program. If the "evil bit" of the packet is set, the eBPF program will extract the label information and store the destination port in a BPF map, `inLabels (dport->[Label,TrackerID])`. Then, the eBPF program removes the DIFC packet header and resets the "evil bit", restoring its original form for further kernel network stack processing. To identify which process receives the labeled packet, the eBPF program monitors processes that invoke a system call to receive a network connection. When a process accepts a connection, the eBPF program looks up the destination port in `inLabels`. If there is a match, the eBPF program will extract the label information and the process ID (PID) of the receiving process, and store the information in another BPF map, `pidLabels (PID->[Label,TrackerID])`.

❷ *Tracing intra-host flows:* The second hook tracks the propagation of DIFC labels between processes and files during intra-host activities through a *data provenance mechanism*. This captures the chain of activities from the process that initially receives the labeled network flow to the process responsible for sending out network traffic. The eBPF program monitors system calls related to process creation, and extracts the PID and the parent PID of the newly created process. Using the parent PID as a lookup key, the eBPF program retrieves the associated label from `pidLabels` and propagates it to the child PID and updates `pidLabels`. If a process is terminated, the eBPF program will remove the entry from `pidLabels`, ensuring that the same PID can be reused. For file operations, the eBPF program maintains a BPF map, `fileLabels (Inode->[Label,TrackerID])`, that associates file inodes with their respective labels. When a new file is created, it is assigned the same label as the creating process, and `fileLabels` is updated. This ensures that the file inherits the appropriate label and aligns with the security context of the creating process. During a file read, the eBPF program retrieves the file's label from `fileLabels` and uses it to update the label of the process (by updating `pidLabels`) that performs the read operation. When a file is deleted, the eBPF program removes the entry from `fileLabels` so that the inode can be reused. Tracking file activities helps identify attackers who may save stolen information in files and exfiltrate it later.

❸ *Altering outgoing packets:* The third hook modifies the outgoing packets by incorporating the propagated DIFC labels. When a process invokes a system call to send a network message, the eBPF program is triggered to search for the PID in `pidLabels`. Once the sending process is identified, the process's label and the source port are stored in a new BPF map, `outLabels (sport->[Label,TrackerID])`. To match outgoing packets, we load the eBPF program into the TC (traffic control) Egress. When a packet exits, the eBPF program checks whether the packet's source port has been marked in `outLabels`. If a match is found, the eBPF program will prepare the corresponding DIFC label in a DIFC packet header and insert the header into the outgoing packet.

BPF maps persistence. As BPF maps reside in the kernel space, they are not persistent across eBPF programs reloading or system reboots. This can cause problems when the

attacker performs file activities. Though the file data persists in the filesystem, the BPF maps can be lost, resulting in inaccurate label propagation.

To keep BPF maps persistent across eBPF programs reloading, our host agent *mounts* the eBPF virtual filesystem to the kernel memory. This allows the eBPF programs to pin their maps to the eBPF virtual filesystem by creating a file descriptor that points to these BPF maps. This file descriptor is linked to a specific pathname in the eBPF filesystem. As a result, the kernel will retain the BPF maps even if the referencing eBPF program is unloaded, as the corresponding file descriptor will keep pointing to the BPF maps.

To further persist BPF maps across system reboots, our host agent *migrates* the BPF maps to the permanent filesystem on the host machine. When the host agent detects the `kernel_restart` or `kernel_power_off` system events, it immediately migrates the BPF maps to a backup file in the permanent filesystem before the system reboots. After the system reboots, the host agent repopulates the BPF maps with the entries from the backup file. This repopulation occurs only once before the host agent resumes its functions upon reboot. Our host agent can also migrate BPF maps in case of a system crash, by detecting abnormal terminations of critical processes using the `process_exit` hooks.

5.5. Distributed Multi-Switch Deployment

P4CONTROL leverages the distributed nature of networks to optimize the deployment of in-network policies in the switches. A naive approach would install identical policies on every switch, which wastes space on switches that would never match those policies. In contrast, P4CONTROL places policies only on switches that are likely to see the matching traffic. This is achieved by placing each policy in the switch that is directly connected to the *destination* address defined in the policy, similar to the setup of distributed firewalls. Once a flow is matched and is allowed to pass, the remaining switches only need to forward its packets, ensuring strict security and consistency of policy enforcement. We choose the destination address instead of the source address because otherwise, an attacker could bypass the policies by using different hosts. With this design, we can significantly reduce the storage overhead and minimize the unnecessary latency from re-matching the same flow.

P4CONTROL offers seamless integration into the existing network infrastructure that uses programmable switches, eliminating the need for installing additional middleboxes while offering minimal disruption to the network performance. The central management of these switches by the control plane ensures up-to-date policy installation and simplifies maintenance. Coordinating a distributed defense as a single entity in large infrastructures is complex. Network segmentation [50] addresses this challenge by dividing the network into distinct segments, each governed by its specific set of policies. P4CONTROL can be a pivotal facilitator in this design by configuring switches within each segment to manage their respective DIFC tags and policies. This configuration provides fine-grained control over individual segments, enhancing defense capabilities by accommodating a larger number of DIFC tags in the network.

Primitive Actions

A ::= label_host(ip, label) | label_file(ip, file_path) | drop | allow | reroute(port) | modify(header) | alert | declassify(tags) | endorse(tags)

Expressions

E ::= header_field | var

Predicates

P ::= match(P && P) | E op E | !P

Policies

C ::= A | if P then C | (C|C)

Operations

$op \in$ {==, >=, <=, contains}

Fig. 7: Syntax of NETCL

6. DIFC Policy Framework

P4CONTROL provides an expressive DIFC policy language, named Network Control Language (NETCL), for specifying diverse DIFC policies to counter different attack scenarios. These policies are enforced in priority order and can be dynamically updated at runtime. Despite multiple domain-specific languages proposed for network management [51], [52] and network security [32], [37], [53], [54], none of them are designed for network-level DIFC policies.

6.1. Expressive Policy Syntax

Fig. 7 illustrates the syntax of NETCL, which is inspired by previous works [32], [37] that adapt NetCore [52] (an SDN programming language) for network defenses.

NETCL provides two labeling functions to initialize DIFC labels: `label_host(ip, label)` assigns a label to a specific host's IP address, initializing all existing processes and files with the host's label; `label_file(host_ip, file_path)` assigns a unique `TrackerID` to a file on a host for fine-grained tracking of sensitive files. These functions can be used during initial deployment or subsequent stages when new hosts or files need labeling.

A NETCL policy consists of a flow-matching predicate and an action. Various *patterns* are provided to match a network flow based on the source and final destination hosts on a cross-host path, the DIFC tags, etc. Expressions (`E`) can represent constants (`var`) such as IP addresses and DIFC tags, as well as DIFC or IP packet header fields (`header_field`) such as `dst_ip` and `pkt_label`. `TrackerID` can be represented by the location of the tagged file (i.e., `file_path@host_ip`). Predicates (`P`) are built over expressions with comparison operations (`E op E`), which are used to match network flows and trigger actions. The keyword `contains` checks a subset of DIFC tags in the DIFC packet header of a network flow.

NETCL provides multiple primitive *security actions*. The `drop` action discards a flow at the switch. The `allow` action forwards a flow based on the configured forwarding table. The `reroute(port)` action redirects suspicious traffic to a predefined destination, such as a logging server or a deep packet inspection (DPI) system, for further scrutiny or processing. The `modify(header)` action uses the programmable parser of the switch to modify the packet header. For instance, P4CONTROL can reset specific packet headers (e.g., IP options and TTL) which may be used as a covert channel for data exfiltration. The `alert` action serves as a detection mechanism, generating alerts to notify the network administrator when a suspicious flow is detected.

NETCL also provides two *privileged actions*. The `declassify(tags)` action removes specified tags, allowing sensitive data declassification. The `endorse(tags)` action adds designated tags, endorsing the flow's integrity. Additionally, the endorsement action allows inserting tags for flows originating from external addresses, where no host agent is installed. This allows P4CONTROL to regulate information flows from external addresses within the network.

6.2. Efficient Policy Compilation

To enforce user-defined NETCL policies in the data plane, P4CONTROL employs an efficient compiler to compile and execute NETCL policies in the switch. The label initialization statements are interpreted, and the switch uses its hardware packet generator to send a control packet containing the DIFC label to the respective host agent.

For NETCL matching policies, it is essential to develop an efficient compilation strategy to counter the rapidly changing behaviors of attackers. When the attacker changes the strategies, the matching patterns must be updated accordingly, and the updated policy must be quickly recompiled and pushed to the switch. A naive compilation strategy would compile a NETCL policy into a P4 program to run in the switch. However, this approach would require reloading the P4 program every time a policy changes, which would interrupt the network traffic and cause significant disruption.

To improve defense agility, P4CONTROL employs an efficient compilation mechanism that supports *dynamic update* of NETCL policies without interrupting traffic. Our idea is to compile NETCL policies into the corresponding *switch configurations*, which are a set of parameters that define a switch's operations, including match-action table entries for packet header matching, associated actions, and policy priority levels. These switch configurations then insert in-network policies into their respective match/action tables within the switch. Note that in this design, we still need to create a P4 program to specify all the logic to parse customized packet headers and define match/action tables. However, this P4 program does *not* contain specific match/action rules, and needs to be compiled by the P4 compiler (different from the NETCL compiler) and loaded into the switch *only once*. After the compilation, the switch configurations are passed to the switch daemon in the control plane. Whenever the NETCL policies change, the NETCL compiler generates new switch configurations, and the switch daemon updates the match/action tables by adding or removing in-network policies accordingly. This mechanism allows the control plane to seamlessly add or remove in-network policies.

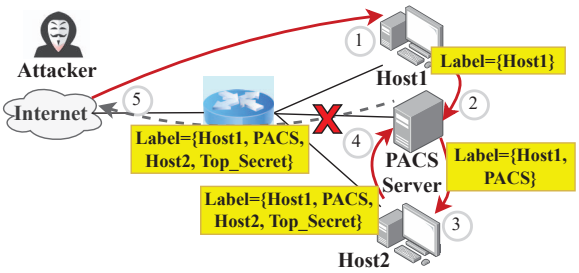


Fig. 8: Preventing data exfiltration

6.3. NETCL Policy Examples

We now present examples of NETCL policies against various attack scenarios to show NETCL's expressiveness.

Scenario 1: Preventing data exfiltration. Fig. 8 illustrates a real-world data leakage incident against a hospital network [55]. The attacker first compromises `Host1`'s web browser to get into the internal network, aiming to exfiltrate sensitive data from `Host2`. The network has a picture archiving and communication system (PACS) server that is less secure and allows widespread data sharing. The firewall blocks direct connections from `Host1` to `Host2` and from `Host2` to the external network. To bypass the firewall, the attacker uses the PACS server as a stepping stone to reach `Host2` and then moves the data from `Host2` to `PACS` and ultimately to the external network.

```
1  # Initialize labels
2  label_host(ip=Host1, label={Host1})
3  label_host(ip=Host2, label={Host2, Top_Secret})
4  label_host(ip=PACS, label={PACS})
5
6  # Drop network flows containing Top_Secret data
7  if match(pkt_label contains Top_Secret &&
       dst_ip==external_network) then drop
8
9  # Allow traffic between hosts and PACS server
10 if match(src_ip==Host1 && dst_ip==PACS) then allow
11 if match(src_ip==PACS && dst_ip==Host2) then allow
12 if match(src_ip==Host2 && dst_ip==PACS) then allow
13
14 ... # Other policies that allow benign traffic
15 # DROP ALL (default deny)
```

Listing 1: Preventing exfiltration of top-secret data

Listing 1 shows the NETCL policies. We omit additional policies that allow benign traffic. Network flows that do not match any policies are dropped by default. Using DIFC labels and a matching policy, we can protect `Host2`'s data with the `Top_Secret` tag from being leaked, regardless of intermediate hosts. For example, when the attacker attempts to export the secret data from `PACS` to the external network, P4CONTROL detects the presence of the `Top_Secret` tag in the network flow and blocks the flow (Line 7).

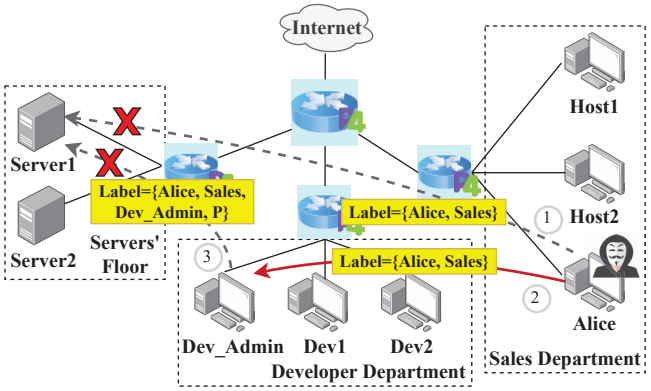


Fig. 9: Preventing unauthorized access while endorsing high-integrity users to access

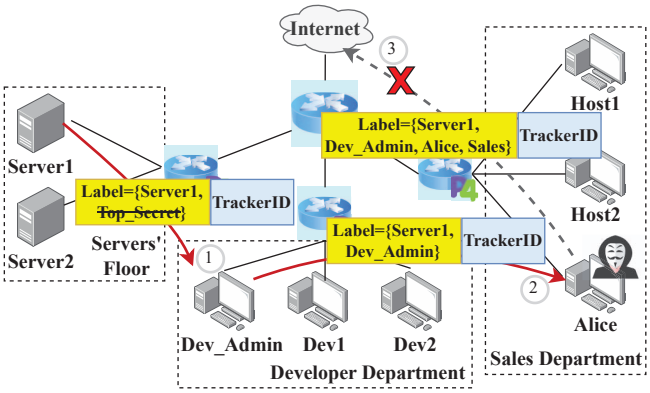


Fig. 10: Fine-grained tracking of sensitive information

Scenario 2: Preventing unauthorized access. Access control is essential in enterprise networks. However, existing solutions fall short of defending against insider threats across hosts. This scenario showcases how we can leverage our in-network endorsement mechanism to prevent unauthorized access. Fig. 9 illustrates an enterprise network where each department is protected by a firewall, only `Alice` can access both the sales and developer resources, and only `Dev_Admin` has permission to access the Servers' Floor. Alice, as an insider attacker, can abuse her permissions and use a zero-day vulnerability to compromise `Dev_Admin` to gain further access to the Servers' Floor.

```
1  # Initialize labels
2  label_host(ip=Sales_Dept, label={Sales})
3  label_host(ip=Alice, label={Alice, Sales})
4  label_host(ip=Dev_Admin, label={Dev_Admin})
5
6  # Endorse network flows (add tag) from Dev_Admin
7  if match(src_ip==Dev_Admin && dst_ip==Servers_Floor)
       then endorse({P})
8
9  # Only allow network flows with the integrity tag P
10 if match(src_ip==Sales_Dept && dst_ip==Servers_Floor)
       then drop
11 if match(pkt_label contains P &&
       dst_ip==Servers_Floor) then allow
12
13 ... # Other policies that allow benign traffic
14 # DROP ALL (default deny)
```

Listing 2: Endorsing users to access protected resources

Listing 2 shows how we can prevent unauthorized insiders from accessing the Servers' Floor while only endorsing `Dev_Admin` for access. We only allow `Dev_Admin` to access the servers, by adding the tag `P` to flows originating from `Dev_Admin` (Line 7) and checking the presence of the tag `P` (Line 11). Even if Alice exploits a vulnerability in `Dev_Admin` (②) to acquire the needed tag, the policy at Line 10 will detect the `Sales` tag in the network flow and block Alice from connecting to the servers (③).

Scenario 3: Fine-grained tracking of sensitive information. Fig. 10 illustrates the situation where fine-grained tracking is needed to further restrict the propagation of declassified files. A protected file on `Server1` is declassified to `Dev_Admin` for sharing within the company's internal network. However, Alice may profit from gaining early access to confidential information and leaking the file to the external network without the permission of the company.

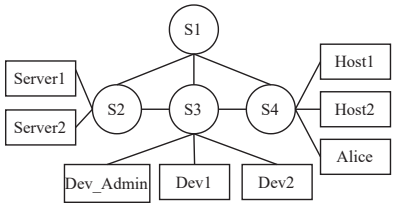
```
1  #Initialize labels and TrackerID for the sensitive file
2  label_file(ip=Server1, file=/server1/sensitive_file)
3  label_host(ip=Server1, label={Server1, Top_Secret})
4  label_host(ip=Dev_Admin, label={Dev_Admin})
5  label_host(ip=Alice, label={Alice, Sales})
6
7  #Declassify Top_Secret (remove tag) data to Dev_Admin
8  if match(src_ip==Server1 && dst_ip==Dev_Admin) then
       declassify({Top_Secret})
9
10 # Prevent the tainted file from leaving the network
11 if match(tracker_id==/server1/sensitive_file@Server1
       && dst_ip==external_network) then drop
12
13 # Prevent Top_Secret data from leaving Server1
14 if match(pkt_label contains Top_Secret && dst_ip==any)
       then drop
15
16 ... # Other policies that allow benign traffic
17 # DROP ALL (default deny)
```

Listing 3: Preventing exfiltration of declassified information

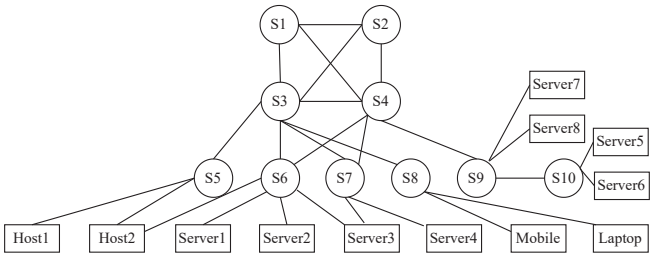
Listing 3 shows how we can track the propagation of sensitive information and prevent the information from being leaked to the external network. We assign a unique `TrackerID` tag to the sensitive file being tracked (Line 2). `Top_Secret` files are protected and cannot leave `Server1` without explicit declassification (Line 14). When P4CONTROL declassifies this file for `Dev_Admin`, it removes the `Top_Secret` tag (Line 8) but retains the `TrackerID` with the network flow (①). After Alice acquires a copy of the file (②) and attempts to leak it, the `TrackerID` persists with her outgoing network flow (③), enabling P4CONTROL to block the flow (Line 11).

7. Evaluation

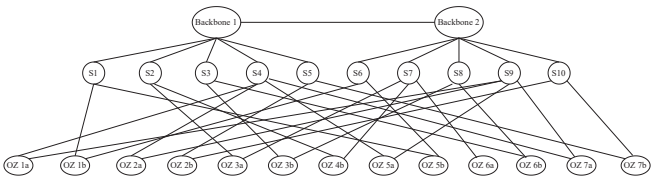
In this section, we aim to answer several key research questions on the defensive effectiveness and cover-



(a) Our example enterprise network



(b) Cisco enterprise network



(c) Stanford backbone network

Fig. 11: Three network topologies (*S* indicates switches)

age, scalability, system capacity, and system overhead of P4CONTROL, through extensive evaluations.

(RQ1) How effective is P4CONTROL in defending against various types of cross-host attacks?

(RQ2) How well does P4CONTROL scale with real-world workloads and network topologies?

(RQ3) What is the capacity of P4CONTROL and how does P4CONTROL impact the network performance and the host performance?

(RQ4) How does P4CONTROL compare with existing SDN-based defenses?

(RQ5) How does P4CONTROL's distributed deployment further optimize its efficiency and scalability?

Implementation and deployment. We implemented a prototype of P4CONTROL in ∼3,200 lines of P4, C, and Python code. This includes the NETCL compiler, eBPF programs, switch program, and switch control plane functions. We deployed P4CONTROL on a testbed of a physical Wedge 100BF-32X Tofino P4 switch with 32×100 Gbps ports. The testbed setting is similar to that of existing P4 security works [33], [36]. Our experiments run on three Dell R420 servers, each equipped with an Intel Xeon E5-2430 CPU running at 2.20 GHz, 64 GB RAM, and Ubuntu 20.04.

TABLE I: Network topologies

Topology	# Hosts	# Switches	Details
Example enterprise	8	4	Our example enterprise topology (Fig. 11a)
Cisco	14	8	Cisco enterprise network (Fig. 11b)
Stanford	56	25	Stanford backbone network (Fig. 11c)

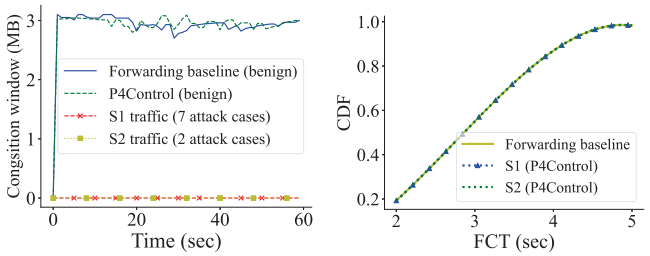
We set up these three servers to act as the protected, intermediate, and attacker hosts, respectively. Additionally, we deployed P4CONTROL in three representative enterprise topologies, including our example enterprise topology, Cisco enterprise network [54], and Standford backbone network [56] (see Table I and Fig. 11). The topologies are constructed using a packet-level simulation in Mininet, which is integrated with a software P4 switch (i.e., bmv2 [57]) that emulates the behavior of physical switches.

7.1. RQ1: Defense Effectiveness and Coverage

We compare P4CONTROL with two real-world network defense solutions: a firewall (e.g., iptables [3]) and an NIDS (e.g., Snort [4]). We configure these defenses to restrict the protected host from initiating or accepting connections from the external network (i.e., the attacker host), while the intermediate host is allowed to communicate with the other two hosts. This mirrors realistic enterprise setups where defenses shield vital resources from the attacker, with less restrictive policies on other devices for easier access. To quantify the network performance, we measure the *TCP congestion window* using iperf3, which reflects the overall throughput. We also measure P4CONTROL's impact on benign traffic by comparing the *flow completion time (FCT)* under the typical forwarding switch (our baseline) and P4CONTROL defense. Background traffic is generated using the Distributed Internet Traffic Generator (D-ITG) [58].

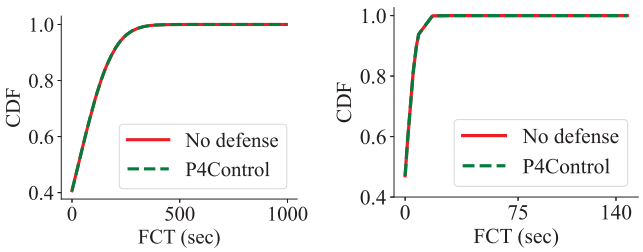
We conduct nine attacks, categorized into two groups: a stealthy insider attacker exfiltrating sensitive information (S1), and an external APT attacker accessing unauthorized resources (S2). In S1, we mimic an insider attacker residing in the protected host and use netcat to transmit a stolen file to the intermediate host. To exfiltrate the file to the attacker host, we use the Data Exfiltration Toolkit (DET) [59] with seven different communication channels, including different protocols and real-world applications (TCP, UDP, DNS, ICMP, HTTP, Twitter, and Gmail). In S2, we use Metasploit from the attacker host to exploit a vulnerability [60] in the intermediate host, escalating privileges and establishing a connection to the protected host. We repeat the attack with the socat tool, which relays the attacker's traffic to the protected host from the intermediate host.

Since the protected host and the attacker host do not communicate directly, both firewall and NIDS fail to block any attack. With P4CONTROL, we label the protected host and the attacker host and create a NETCL policy to block communications between the two hosts. As shown in Fig. 12a, P4CONTROL blocks all nine attacks (i.e., congestion window at 0), ensuring security while maintaining a congestion window of benign traffic similar to the baseline. Also, we observe no significant difference in the cumulative



(a) Congestion window **(b)** Benign traffic FCT

Fig. 12: P4CONTROL blocks all stealthy cross-host attacks while imposing minimal overhead on benign traffic



(a) FCT in LANL **(b)** FCT in DARPA OpTC

Fig. 13: P4CONTROL imposes minimal overhead under real-world enterprise workloads while blocking all attacks

distribution function (CDF) of the FCT of the benign traffic between P4CONTROL and the baseline in Fig. 12b, showing that the defense has a negligible impact on benign traffic.

7.2. RQ2: Scalability in Real-World Scenarios

Scalability with real-world enterprise workloads. We evaluate the scalability of P4CONTROL using the Los Alamos National Laboratory (LANL) Unified Host and Network dataset [29] and the DARPA Operationally Transparent Cyber (OpTC) dataset [28]. The LANL dataset contains benign activities from 17,500 hosts. The DARPA OpTC dataset contains both benign and APT activities (initial compromise, lateral movements, privilege escalations, etc.) across 1,000 hosts. Notably, in LANL, up to 80% of daily network flows in enterprises use TCP protocol, emphasizing the need for efficient defenses that do not impact the sending rate. Also, attackers typically pivot to infect more machines. In DARPA OpTC, an attacker pivoted across 14 hosts beyond its initial comprise, necessitating the need to limit the attacker's reachability. Importantly, a very small portion of these enterprise events are associated with attackers' activities (e.g., only 0.0016% in DARPA OpTC), requiring controls that do not interfere with benign traffic.

We use D-ITG to replay the two workloads to the physical switch and observe network performance during a "no-defense" baseline and under P4CONTROL. As LANL lacks malicious traces, we simulate real cross-host malicious flows (similar to S1 and S2 scenarios) for a comprehensive evaluation. We label the initial victim hosts with a `V` tag and the final target hosts with a `protected` tag. To block the multi-hop malicious access, we create the NETCL policy: **if match**(pkt_label **contains** V && dst_ip==target) **then drop**.

By only appending DIFC labels to the SYN packet in TCP flows, our in-network per-flow decision technique effectively reduces the storage overhead of appending DIFC packet headers (6.4% reduction for 500-byte data packets). Additionally, P4CONTROL blocks all malicious network flows from reaching the target hosts regardless of the number of intermediate hosts compromised. As shown in Fig. 13, the "no-defense" baseline and P4CONTROL have approximately the same average FCTs across all flows in both datasets. This confirms that P4CONTROL scales well with real-world workloads, provides an effective defense against real-world cross-host attacks, and imposes minimal overhead on benign traffic due to its data plane enforcement. Such observation is consistent with our testbed experiments in Section 7.1.

Limiting attacker's reachability. We analyze the reachability of an external APT attacker, who aims to access `Server1` and `Server2` in our enterprise topology in Fig. 11a. Assuming that all hosts are vulnerable, we assess the attacker's reachability to the servers by exploiting each host and pivoting using varying *step-counts*, which refers to the number of intermediate hosts used for pivoting. In one setup, we configure the distributed firewalls as follows: only hosts directly connected to switch `S4` can access `Server1`, only `Dev_Admin` can access `Server2`, and hosts on `S4` are blocked from those on `S3`, except for Alice. In another setup, we run P4CONTROL and assign a unique label to each host.

We run a script that simulates an external attacker performing APT steps (port scanning, exploiting hosts, escalating privileges, pivoting, etc.). The script aims to infiltrate hosts in the network and pivot towards the target servers, until either reaching them or exhausting the allowed step-counts. Table II shows that the distributed firewall can easily detect attempts with step-count of 1, as they are direct accesses. However, as the allowed step-count increases, the number of hosts that can be exploited and ultimately reach the servers increases. In contrast, P4CONTROL blocks all attempts regardless of the number of intermediate hosts.

Attack routes coverage. To further evaluate P4CONTROL's attack routes coverage, we simulate an insider threat within the three enterprise networks in Fig. 11. We select a target machine and an insider attacker attempting to traverse the network to reach the target. A firewall is configured to restrict the target's direct communications to a subset of allowed hosts inside the network. We then run an attacker script that probes all possible routes to the selected target and records the number of successful accesses.

Table III shows (1) the number of potential attack routes under different network sizes and allowed step-counts and (2) attack routes coverage under different numbers of allowed hosts. Out of all potential attack routes to a target, the deployed firewall may only be able to block some of them. *Attack routes coverage* refers to the proportion of attack routes that are blocked. The more hosts that are allowed

TABLE II: Number of hosts that can reach `Server1` and `Server2` with different step-counts in the example enterprise network in Fig. 11a

Defense system	Step-count	Server1	Server2
Distributed firewall	1	3	1
	2	7	3
	3	7	7
P4CONTROL	1	3	1
	2	3	1
	3	3	1

TABLE III: Number of attack routes in each network

Topology	Step-count	Total routes	Allowed hosts	Attack routes coverage	
				Distributed firewall	P4CONTROL
Example enterprise	6	5,040	1	85%	**100%**
	5	2,520	2	70%	
	4	840	3	57%	
	3	210	4	42%	
	2	42	5	28%	
Cisco	5	154,440	2	84%	**100%**
	4	17,160	4	67%	
	3	1,716	6	53%	
	2	156	8	38%	
Stanford	4	8,185,320	10	81%	**100%**
	3	157,410	20	63%	
	2	2,970	30	45%	

direct access to the target, the lower the percentage of attack routes the firewall will be able to block. This is common in enterprise networks, where a single user's access to multiple domains increases the potential attack routes. In contrast, P4CONTROL covers 100% of attack routes and limits the target's access to the allowed hosts only. This holds no matter whether the attacker leverages the allowed host as a stepping stone to reach the target or not. Notably, this protection is achieved using only a single NETCL policy that blocks the attacker's label from reaching the target.

Maximum number of active connections. We leverage the P4 compiler to assess the maximum number of active connections that P4CONTROL can handle, as it rejects a program if it consumes more memory than available resources. By progressively increasing the number of active connections, we find that P4CONTROL can support more than 220K concurrent active connections. This surpasses the number of active connections found in Facebook frontend clusters, which ranges from 10K to 100K [61].

7.3. RQ3: System Capacity and Overhead

In-network policies and DIFC tags supported. We measure P4CONTROL's capacity in handling various DIFC packet header sizes and in-network policies within a single switch, by progressively increasing the header size and policy count until the P4 compiler rejects the program. With a single flow matching table, P4CONTROL can maintain less than 1K policies due to the limited TCAM size. In contrast, our multi-table flow matching technique enables P4CONTROL to maintain ∼12K unique policies, even with the largest 32-byte DIFC packet header that supports 256 tags. Table IV shows the number of in-network policies as the DIFC packet header size grows. For comparison, the Stanford backbone network requires ∼1,500 access control list (ACL) policies [56]. This indicates that a single switch deployment can accommodate many more policies than those typically used in large real-world networks.

TABLE IV: Number of supported in-network policies with different DIFC packet header sizes

DIFC packet header size	# DIFC tags	# In-network policies
1-byte	8	140K
2-byte	16	140K
4-byte	32	72K
8-byte	64	48K
10-byte	80	36K
16-byte	128	24K
32-byte	256	12K

As for tags, the single switch capacity for 256 tags largely exceeds the minimum access control requirement (16 sensitivity classifications and 64 categories) by the U.S. Department of Defense [45]. This can be further increased through network segmentation as discussed in Section 5.5. Additionally, the latest switch models (e.g., Tofino 2/3 [46], [47]) have 3× more resources than our current model, supporting a larger number of tags and in-network policies.

Switch resource utilization. Fig. 14 shows how the switch resource utilization varies with the number of active connections, the tag size, and the number of policies. In the single-switch deployment, with 220K active connections, 256-bit tag size, and 12K policies, the resource utilization is 26% SRAM, 25% TCAM, 5.7% VLIW, 2.1% meter ALU, and 8.9% hash units. When we reduce the tag size to 80 bits while keeping the same number of active connections and policies, the TCAM utilization decreases to 8.3%, while other resources remain roughly unchanged. This indicates that the tag size has a significant impact on the TCAM utilization, as matching larger DIFC packet headers requires more TCAM. We repeat the experiment but with 100K active connections, and the SRAM utilization decreases to 14.1% while other resources remain unchanged. This indicates that the more active connections the switch handles, the more SRAM it requires for maintaining decisions of established connections.

We further compare the resource utilization of single-switch deployment and multi-switch deployment. We use 220K active connections, 256-bit tag size, and 12K policies, and deploy P4CONTROL on three switches to distribute the policies. As shown in Fig. 14, the TCAM utilization noticeably drops from 25% to 8.5% on each switch, as only 4K policies are maintained within each switch.

Impact on network throughput and latency. We evaluate the impact of P4CONTROL on the network throughput and latency. We compare different P4CONTROL actions with a forwarding baseline program (Fwd). We use the on-switch hardware packet generator, which can generate 100 Gbps (per-port) traffic for stress testing. As shown in Fig. 15, P4CONTROL achieves a throughput of 99.9 Gbps, maintaining the highest performance of our switch. Also, it introduces a latency overhead of 100-110 ns when compared with

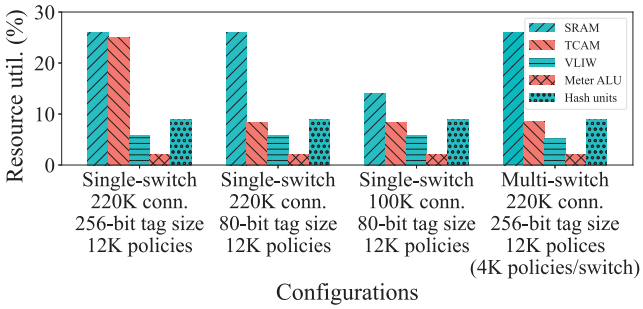


Fig. 14: Switch resource utilization in different settings

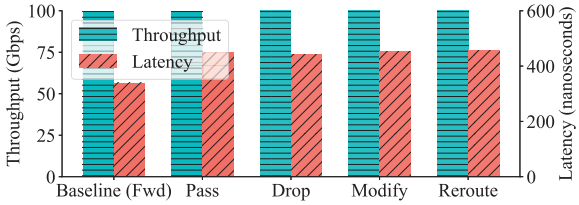


Fig. 15: Throughput and latency under P4CONTROL compared to the forwarding baseline

the baseline, which is negligible since the RTT in typical enterprise networks is in the order of milliseconds [62].

Host agent overhead. We evaluate the host agent's overhead on a Linux server (kernel version 5.15.0) by measuring the average *additional latency* of the eBPF programs over 10K runs. For network ingress and egress eBPF programs, we generate 10K labeled network flows with 10 packets each, and measure the runtime of these two programs. Table V shows that the host agent adds an overhead of 1-7 ms per system call, which is negligible compared to the total runtime of most system calls. Notably, for read/write operations, the eBPF programs take a similar time as the actual system call to record the label in the corresponding BPF map. However, they execute after the system call returns, thereby not blocking the completion of system call operations.

We further measure the storage overhead of maintaining DIFC labels using BPF maps. For `pidLabels`, as there are only 2^{15} process IDs typically available in a Linux system, our table consumes a size of 1.3 MB, which is negligible considering the current abundant storage in systems. As for `fileLabels`, the default setting in Linux allocates one inode for every 16 KB of space. Therefore, in a 1 GB filesystem, `fileLabels` consumes 2.6 MB to maintain the labels for the inodes that can be assigned, which only takes 0.25% of the whole storage in the filesystem.

7.4. RQ4: Comparison with SDN-Based Solution

We compare P4CONTROL with PivotWall [25], an OpenFlow SDN-based IFC approach. PivotWall propagates taint tags between system entities within hosts. When two hosts communicate, the sender host adds the taint tag to outgoing packets and sends "control messages" with the taint

TABLE V: Host agent overhead (in ms)

	execve	clone	TC egress	XDP ingress	read	write
System call time	466	266	59	36	8	9
P4CONTROL overhead	+6	+5	+0.7	+0.3	+5	+6

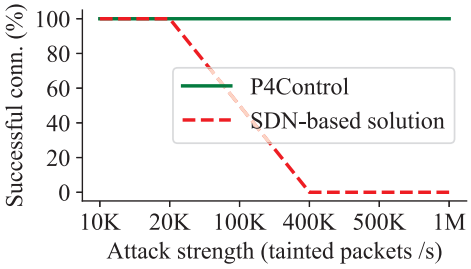


Fig. 16: Control plane saturation attack against an SDN-based solution (PivotWall [25]) and P4CONTROL

information to the SDN controller. The controller maintains a local graph of the taint propagation path. Upon receiving the "control messages", the controller matches the incoming tainted network flow and the local graph with the policies. Then, the controller installs the corresponding decision in the switch. As PivotWall is not open-sourced, we implement it as an SDN application. In our testbed, we set up a Floodlight SDN controller on one server and configure another one with OpenvSwitch for OpenFlow communication. We use a third server to generate traffic for evaluation.

Precise information confinement. PivotWall relies solely on the taint information of system entities and lacks precise information confinement. It also lacks safe controls to declassify or endorse data. Furthermore, it uses a modified Linux kernel to track information within hosts, which requires intensive kernel changes. In contrast, P4CONTROL supports a full DIFC model for precise information confinement via DIFC labels, and has mechanisms to safely move data between different compartments. This level of fine-grained control cannot be achieved in PivotWall with coarse-grained tracking. In addition, P4CONTROL can be seamlessly integrated with hosts using eBPF without kernel modifications, largely simplifying the agent deployment.

Defense responsiveness. We compare the time taken by each defense to install the decision after receiving a tainted packet. With PivotWall, the "control messages" must be routed to the control plane for matching, incurring a *round-trip delay* before a decision is pushed to the switch. Our measurement shows that this process takes between 8 ms to 3 seconds until a decision is pushed, depending on network traffic and graph lookup time. In comparison, P4CONTROL performs the flow matching in the *data plane*, achieving a much smaller delay of less than 500 ns.

Control plane saturation attacks. Centralized SDN-based solutions introduce a single point of failure, potentially causing performance bottlenecks and security vulnerabilities. We simulate an attacker that overwhelms the link between the switch and the controller by initiating a large number of connections. Fig. 16 shows how PivotWall's

central controller struggles to process new connections when the attack strength exceeds 100K tainted packets/s. At an attack strength of 1M packets/s, *99% of the legitimate connections are dropped*, making PivotWall ineffective. Since P4CONTROL examines packets entirely in the data plane, it maintains a stable performance and installs 100% of the connections during the attack.

7.5. RQ5: Distributed Storage Optimization

Our distributed multi-switch deployment significantly reduces the storage overhead of in-network policies. Applied to our enterprise network in Fig. 11a, with 100 in-network policies per host (amounting to 800 unique policies), P4CONTROL distributes the policies across the three switches that are directly connected to the hosts. Such distribution reduces each switch's storage overhead by an average of $\sim$66%. This allows for efficient utilization of the switch resources and ensures consistent policy enforcement. Also, by performing the policy matching on a single switch, P4CONTROL only incurs an additional latency of $\sim$110 ns, which is negligible, regardless of the number of switches that a network flow traverses on its path to its destination.

8. Discussion

Limitations of host agent. While eBPF provides a secure sandbox environment for running user-defined programs in the kernel, there are still some potential vulnerabilities with the current technology. By exploiting existing vulnerabilities within eBPF through malicious code [63], an attacker could execute arbitrary memory reads and writes, compromising the integrity of our BPF maps. Fortunately, there have been efforts to harden eBPF through improved safety verification of eBPF programs [64] and fine-grained BPF privileges [65], which can enhance the security of our host agent.

Robustness against integrity poisoning. NETCL policies can prevent integrity poisoning attacks that target DIFC systems. In such attacks, malicious hosts with low integrity levels may attempt to connect to benign hosts with high integrity levels, lowering their integrity and restricting their access to high-integrity resources. Through NETCL policies, the network administrator can assign appropriate integrity levels to hosts or domains, block low-integrity network flows from communicating with high-integrity hosts, and selectively endorse valid flows only from hosts with access permissions. With flexibility and expressiveness of NETCL, the network administrator can safeguard communications across different integrity levels and block poisoning attempts.

Policy deployment in dynamic scenarios. P4CONTROL reduces the switch resource utilization by distributing policies to multiple switches. However, this static deployment can face challenges in dynamic network environments, such as network topology changes or switch failures, which require policy reallocation. Moreover, the switch has limited resources that must be shared with other data plane applications. The available resources may vary dynamically due to policy updates and the loading/unloading of other applications, which further complicates the policy deployment. P4CONTROL can benefit from an online policy deployment strategy that dynamically reallocates policies while ensuring enforcement consistency and balancing resource usage.

Zero trust architecture. Zero trust (ZT) is an evolving set of security paradigms that assume no implicit trust of any user account or asset based on their physical or network location or ownership. Instead, ZT requires persistent verification of every interaction with the least privileges granted [31]. It is a radical shift from the traditional "castle-and-moat" network security model that relies on perimeter defenses and implicit trust inside the network. Motivated by the U.S. White House issued Executive Order EO-14028 [66] and Memo M-22-09 [67], ZT has recently gained wide attention. P4CONTROL's ability for fine-grained least-privilege network access control via in-network DIFC aligns with ZT principles.

P4CONTROL can be further extended to realize ZT goals in enterprise networks. More types of complex security and integrity policies that incorporate behavioral host attributes can be designed. These attributes can be collected by our eBPF-based host agent and analyzed by an intelligent data plane that runs a machine learning model (e.g., decision tree [68]). This behavioral analysis enables continuous assessment of user and device profiles for adaptive access control. Being integrated into the existing network infrastructure with minimal modifications and overhead, P4CONTROL transforms the network into a defense backbone, serving as a valuable component for implementing a ZT architecture in enterprise networks.

9. Related Work

Programmable switches. Recent works proposed to offload networking tasks to the data plane [69], [70]. In addition, many works leverage data plane programmability to develop security primitives that run at line rate [32]–[37]. Unlike P4CONTROL, none of these works focus on real-time prevention of sophisticated cross-host attacks.

DIFC. In Section 2, we reviewed existing DIFC works in detail, discussed their limitations, and explained why they are unsuitable for our goal. P4CONTROL is the first work that realizes DIFC at the network level at line rate.

System auditing. Prior works proposed to collect system audit logs of system calls and construct system provenance graphs to aid attack investigation. These works, such as [6]–[9], proposed different techniques for comprehensive system provenance analysis. Other works discussed cross-host attacks by associating host-level provenance [11]–[14], which primarily target post-attack forensic investigation instead of real-time attack prevention. P4CONTROL differs from all these system-level defenses in proposing a new paradigm of network-level APT defenses using programmable data planes with line-rate defense performance.

Recent works also proposed domain-specific languages to query attack behaviors from system audit logs [10],

[71]–[73]. However, these languages are not designed for network-level DIFC policies and are unable to express complex secrecy and integrity policies either.

SDN. Recent works proposed SDN-based solutions to extend packets with taint tags derived from host-level information [25]. However, their centralized design incurs high network latency and exposes additional attack vectors to the control plane. P4CONTROL leverages data plane programmability to address these issues, augmenting the defense with line-rate performance and minimal overhead.

10. Conclusion and Future Work

We proposed P4CONTROL, a network defense system for preventing cross-host attacks in real time. P4CONTROL employs a novel in-network DIFC mechaism based on programmable switches and eBPF, and offers an expressive policy framework for specifying DIFC policies. P4CONTROL is effective against various cross-host attacks while maintaining line-rate performance with minimal overhead.

There are a few future directions that are worth exploring. First, P4CONTROL's DIFC enforcement scope can be extended to include the confinement of information within hosts. This can be achieved by extending the functionalities of our host agent, similar to previous OS-level DIFC systems, but with minimal kernel modifications and host overhead offered by eBPF. Second, we can design a dynamic multi-switch deployment strategy using online optimizations, which can optimize the policy deployment based on available switch resources and adapt to dynamic network changes. Third, we can extend P4CONTROL to implement a zero trust architecture, with more complex secrecy and integrity policies that incorporate behavioral host attributes.

Acknowledgement

We would like to thank the anonymous reviewers and our shepherd for their constructive comments and suggestions. This work is supported in part by the Commonwealth Cyber Initiative (CCI). Any opinions, findings, and conclusions made in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] CrowdStrike, "Lateral movement," 2023. [Online]. Available: https://www.crowdstrike.com/cybersecurity-101/lateral-movement/

[2] Colonial, "The great data heist: The 21st century's biggest data breaches," 2020. [Online]. Available: https://www.colonialsurety.com/the-great-data-heist-the-21st-centurys-biggest-data-breaches-blog/

[3] "The netfilter.org "iptables" project." [Online]. Available: https://www.netfilter.org/projects/iptables/index.html

[4] "Snort." [Online]. Available: https://www.snort.org/

[5] "Ipv6 flow label specification ietf rfc6437." [Online]. Available: https://www.rfc-editor.org/rfc/rfc6437

[6] S. Ma, X. Zhang, and D. Xu, "Protracer: Towards practical provenance tracing by alternating between logging and tainting," in *NDSS*, 2016.

[7] S. T. King and P. M. Chen, "Backtracking intrusions," *SIGOPS Oper. Syst. Rev.*, p. 223–236.

[8] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. Stoller, and V. Venkatakrishnan, "SLEUTH: Real-time attack scenario reconstruction from COTS audit data," in *USENIX Security*, 2017, pp. 487–504.

[9] P. Fang, P. Gao, C. Liu, E. Ayday, K. Jee, T. Wang, Y. F. Ye, Z. Liu, and X. Xiao, "Back-propagating system dependency impact for attack investigation," in *USENIX Security*, 2022, pp. 2461–2478.

[10] P. Gao, X. Xiao, Z. Li, F. Xu, S. R. Kulkarni, and P. Mittal, "AIQL: Enabling efficient attack investigation from system monitoring data," in *USENIX ATC*, 2018, pp. 113–125.

[11] Y. Ji, S. Lee, E. Downing, W. Wang, M. Fazzini, T. Kim, A. Orso, and W. Lee, "Rain: Refinable attack investigation with on-demand inter-process information flow tracking," in *CCS*, 2017, p. 377–390.

[12] Y. Ji, S. Lee, M. Fazzini, J. Allen, E. Downing, T. Kim, A. Orso, and W. Lee, "Enabling refinable Cross-Host attack investigation with efficient data flow tagging and tracking," in *USENIX Security*, 2018, pp. 1705–1722.

[13] A. Gehani and D. Tariq, "Spade: Support for provenance auditing in distributed environments," in *Middleware*, 2012, pp. 101–120.

[14] M. N. Hossain, S. Sheikhi, and R. Sekar, "Combating dependence explosion in forensic analysis using alternative tag propagation semantics," in *IEEE S&P*, 2020, pp. 1139–1155.

[15] A. C. Myers and B. Liskov, "A decentralized model for information flow control," *ACM SIGOPS OSR*, pp. 129–142, 1997.

[16] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris, "Information flow control for standard os abstractions," *ACM SIGOPS OSR*, pp. 321–334, 2007.

[17] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazieres, "Making information flow explicit in histar," *CACM*, pp. 93–101, 2011.

[18] P. Efstathopoulos, M. Krohn, S. VanDeBogart, C. Frey, D. Ziegler, E. Kohler, D. Mazières, F. Kaashoek, and R. Morris, "Labels and event processes in the asbestos operating system," *SIGOPS OSR*, p. 17–30, 2005.

[19] N. Zeldovich, S. Boyd-Wickizer, and D. Mazières, "Securing distributed systems with information flow control," in *USENIX NSDI*, 2008, p. 293–308.

[20] W. Cheng, D. R. Ports, D. Schultz, V. Popic, A. Blankstein, J. Cowling, D. Curtis, L. Shrira, and B. Liskov, "Abstractions for usable information flow control in aeolus," in *USENIX ATC*, 2012, pp. 139–151.

[21] I. Papagiannis and P. Pietzuch, "Cloudfilter: Practical control of sensitive data propagation to the cloud," in *CCSW*, 2012, p. 97–102.

[22] V. Pappas, V. P. Kemerlis, A. Zavou, M. Polychronakis, and A. D. Keromytis, "Cloudfence: Data flow tracking as a cloud service," in *RAID*, 2013, pp. 411–431.

[23] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM CCR*, p. 87–95, 2014.

[24] "ebpf official website." [Online]. Available: https://ebpf.io/

[25] T. OConnor, W. Enck, W. M. Petullo, and A. Verma, "Pivotwall: Sdn-based information flow control," in *SOSR*, 2018, pp. 1–14.

[26] J. Sonchack, D. Loehr, J. Rexford, and D. Walker, "Lucid: A language for control in the data plane," in *SIGCOMM*, 2021, p. 731–747.

[27] "Intel® tofino™ programmable ethernet switch asic." [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html

[28] A. W. and O. M., "Operationally transparent cyber datase," 2020. [Online]. Available: https://github.com/FiveDirections/OpTC-data

[29] M. J. M. Turcotte, A. D. Kent, and C. Hash, *Unified Host and Network Data Set*. World Scientific, 2018, pp. 1–22.

[30] "P4Control source code." [Online]. Available: https://github.com/peng-gao-lab/p4control

[31] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, *Zero trust architecture. National Institute of Standards and Technology*, 2019.

[32] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric ddos attacks with programmable switches," in *NDSS*, 2020.

[33] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, and V. Sekar, "Jaqen: A High-Performance Switch-Native approach for detecting and mitigating volumetric DDoS attacks with programmable switches," in *USENIX Security*, 2021, pp. 3829–3846.

[34] J. Xing, W. Wu, and A. Chen, "Ripple: A programmable, decentralized Link-Flooding defense against adaptive adversaries," in *USENIX Security*, 2021, pp. 3865–3881.

[35] H. Zhou, S. Hong, Y. Liu, X. Luo, W. Li, and G. Gu, "Mew: Enabling large-scale and dynamic link-flooding defenses on programmable switches," in *IEEE S&P*, 2023, pp. 3178–3192.

[36] J. Xing, Q. Kang, and A. Chen, "NetWarden: Mitigating network covert channels while preserving performance," in *USENIX Security*, 2020, pp. 2039–2056.

[37] Q. Kang, L. Xue, A. Morrison, Y. Tang, A. Chen, and X. Luo, "Programmable In-Network security for context-aware BYOD policies," in *USENIX Security*, 2020, pp. 595–612.

[38] D. E. Bell, L. J. La Padula *et al.*, "Secure computer system: Unified exposition and multics interpretation," *Mitre Corporation Bedford, MA*, 1976.

[39] K. J. Biba, "Integrity considerations for secure computer systems," *Mitre Corporation Bedford, MA*, 1976.

[40] D. E. Denning, "A lattice model of secure information flow," *CACM*, pp. 236–243, 1976.

[41] A. Nadkarni, B. Andow, W. Enck, and S. Jha, "Practical DIFC enforcement on android," in *USENIX Security*, 2016, pp. 1119–1136.

[42] R. Paccagnella, P. Datta, W. U. Hassan, A. Bates, C. Fletcher, A. Miller, and D. Tian, "Custos: Practical tamper-evident auditing of operating systems using trusted execution," in *NDSS*, 2020.

[43] A. Ahmad, S. Lee, and M. Peinado, "Hardlog: Practical tamper-proof system auditing using a novel audit device," in *IEEE S&P*, 2022, pp. 1791–1807.

[44] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proc. IEEE*, pp. 1278–1308, 1975.

[45] *Department of Defense Trusted Computer System Evaluation Criteria (Orange Book)*, 1985, pp. 1–129.

[46] "Intel® tofino 2 12.8 tbps, 20 stage, 4 pipelines." [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/218648/intel-tofino-2-12-8-tbps-20-stage-4-pipelines/specifications.html

[47] "Intel® tofino™ 3 intelligent fabric processors." [Online]. Available: https://www.intel.com/content/dam/www/central-libraries/us/en/documents/2022-05/tofino-3-intelligent-fabric-processors-brief.pdf

[48] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary cams," in *SIGCOMM*, 2005, p. 193–204.

[49] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, "The express data path: Fast programmable packet processing in the operating system kernel," in *CoNEXT*, 2018, p. 54–66.

[50] NordLayer, "What is network segmentation: a complete guide." [Online]. Available: https://nordlayer.com/learn/network-security/network-segmentation/

[51] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," in *ICFP*, 2011, p. 279–291.

[52] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software defined networks," in *USENIX NSDI*, 2013, pp. 1–13.

[53] M. Vallentin, V. Paxson, and R. Sommer, "VAST: A unified platform for interactive network forensics," in *NSDI*, 2016, pp. 345–362.

[54] T. Yu, S. K. Fayaz, M. P. Collins, V. Sekar, and S. Seshan, "Psi: Precise security instrumentation for enterprise networks." in *NDSS*, 2017.

[55] D. Storm, "Medjack: Hackers hijacking medical devices to create backdoors in hospital networks," 2015. [Online]. Available: https://www.computerworld.com/article/2932371/medjack-hackers-hijacking-medical-devices-to-create-backdoors#-in-hospital-networks.html

[56] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *USENIX NSDI*, 2012, pp. 113–126.

[57] "Behavioral model (bmv2)." [Online]. Available: https://github.com/p4lang/behavioral-model

[58] A. Botta, A. Dainotti, and A. Pescapè, "A tool for the generation of realistic network workload for emerging networking scenarios," *Comput. Netw.*, pp. 3531–3547, 2012.

[59] "Det (extensible) data exfiltration toolkit," 2019. [Online]. Available: https://github.com/PaulSec/DET

[60] NIST, "Cve-2004-2687 detail," 2008. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2004-2687

[61] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *SIGCOMM*, 2017, p. 15–28.

[62] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and effective fine-grained tcp retransmissions for datacenter communication," in *SIGCOMM*, 2009, p. 303–314.

[63] MITRE, "Cve-2022-2905," 2022. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-2905

[64] S. Y. Lim, X. Han, and T. Pasquier, "Unleashing unprivileged ebpf potential with dynamic sandboxing," in *Workshop on eBPF and Kernel Extensions*, 2023.

[65] J. Corbet, "Finer-grained bpf tokens." [Online]. Available: https://lwn.net/Articles/947173/

[66] T. W. House, "Executive order on improving the nation's cybersecurity," 2021. [Online]. Available: https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

[67] E. O. of the President, "Moving the u.s. government toward zero trust cybersecurity principles," 2022. [Online]. Available: https://www.whitehouse.gov/wp-content/uploads/2022/01/M-22-09.pdf

[68] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *USENIX Security*, 2023, pp. 6203–6220.

[69] B. Turkovic, F. Kuipers, N. van Adrichem, and K. Langendoen, "Fast network congestion detection and avoidance using p4," in *NEAT*, 2018, p. 45–51.

[70] Y. Li, R. Miao, C. Kim, and M. Yu, "FlowRadar: A better NetFlow for data centers," in *USENIX NSDI*, 2016, pp. 311–324.

[71] P. Gao, X. Xiao, D. Li, Z. Li, K. Jee, Z. Wu, C. H. Kim, S. R. Kulkarni, and P. Mittal, "SAQL: A stream-based query system for real-time abnormal system behavior detection," in *USENIX Security*, 2018, pp. 639–656.

[72] P. Gao, F. Shao, X. Liu, X. Xiao, Z. Qin, F. Xu, P. Mittal, S. R. Kulkarni, and D. Song, "Enabling efficient cyber threat hunting with cyber threat intelligence," in *ICDE*, 2021, pp. 193–204.

[73] T. Pasquier, X. Han, T. Moyer, A. Bates, O. Hermant, D. Eyers, J. Bacon, and M. Seltzer, "Runtime analysis of whole-system provenance," in *CCS*, 2018, pp. 1601–1616.

Appendix A.
Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

A.1. Summary

This paper proposes P4CONTROL, a network defense system capable of controlling flows and preventing cross-host attacks in real time by leveraging programmable switches and eBPF. P4CONTROL creates and propagates DIFC labels of network flows and acts on labeled flows in the data plane according to DIFC rules specified by the network administrator. The authors demonstrate that P4CONTROL is feasible for switch hardware, lightweight on host agents, and effective for various cross-host attacks.

A.2. Scientific Contributions

- Creates a New Tool to Enable Future Science
- Addresses a Long-Known Issue
- Provides a Valuable Step Forward in an Established Field
- Establishes a New Research Direction

A.3. Reasons for Acceptance

1) This paper creates a new tool, P4CONTROL, which enforces DIFC from the network at line rate using programmable switches and eBPF. The approach further enables practical DIFC enforcement by using new networking architectures to implement it.
2) This paper presents a system that solves a well-motivated problem. Technical details on enforcing DIFC in the data plane are interesting and well written.
3) The proposed approach also establishes a new research direction in using programmable data planes to enforce complex integrity and security policies at line rate.