SentinelX: A Lightweight Malicious Traffic Detection System Based on Programmable Switches

Zutao Zhang*‡, Zeyu Luan†‡, Qing Li†§, Zhuyun Qi*†§, Kejun Li†, Yong Jiang*†, Zhenhui Yuan¶
*Tsinghua Shenzhen International Graduate School, China
†Pengcheng Laboratory, China
¶University of Warwick, United Kingdom

***Abstract*—In recent years, programmable switches have emerged as robust platforms for deploying high-performance network services to detect malicious traffic. However, current researches face several challenges: firstly, the flow tables generated by model deployment are cumbersome; secondly, existing unsupervised methods have difficulty handling repetitive traffic; and thirdly, the flow-level inference is coarse-grained and susceptible to attacks. To address these challenges, we propose SentinelX, which offers several advancements. Initially, we design a space-saving multi-level flow table representation method. We then introduce TreeDivider, an innovative model-splitting algorithm that achieves significant space reductions of up to 63.88% after only two subdivisions. Next, we propose DualTree, a hardware-specific unsupervised decision tree utilizing a dual threshold mode, which enhances detection accuracy by approximately 30.24%. Finally, we design a fine-grained method for determining the inference point, boosting the detection rate of bypass attacks by 30.03%. Extensive experiments on the H3C S9830-32H-H1 switch demonstrate that SentinelX can reach 99.99% of the maximum bandwidth of switch ports with nanosecond-level latency, approximately 1.38 times the delay of L3 (network layer) base forwarding.**

***Index Terms*—programmable switches, in-network classification, malicious traffic detection, unsupervised decision tree**

I. Introduction

With the emergence of new network technologies such as 5G, 6G, and the Internet of Things (IoT), network security challenges are also increasing [1]–[3]. Among these challenges, emerging networks impose higher requirements on malicious traffic detection, especially regarding high throughput and low latency. For example, 5G networks can handle a large number of concurrent data streams. As a result, malicious traffic detection should quickly identify and address potential threats without compromising network performance [4], [5]. Additionally, IoT devices rely on real-time data transmission to perform critical tasks, such as vehicle-to-everything (V2X) communication in transportation systems and healthcare data transmission in remote diagnostics. These applications require networks to have low latency characteristics to ensure responsive communications [6], [7].

Programmable switches have become a promising platform for deploying high-throughput and low-latency malicious traffic detection systems. This is because switches often perform packet-forwarding tasks and can handle a large volume of packets. By deploying intelligent malicious traffic detection algorithms on programmable switches at critical network nodes, malicious traffic can be identified and intercepted in real-time as it flows through the network, significantly enhancing the response speed and detection accuracy of the network security system.

Deploying malicious traffic detection systems on programmable switches, however, faces a significant challenge: overcoming the hardware limitations of programmable switches. Due to their limited memory capacity and limited support for floating-point operations, these switches are not suitable for neural networks that require complex computations. Previous studies—such as HorusEye [7], Mousika [8], Netbeacon [9], pForest [10], SwitchTree [11], Flowrest [12]—have adopted concise and effective decision tree-based models for deployment in programmable switches for malicious traffic detection. However, these methods have shortcomings in the following aspects: (i) Scalability, e.g., the lack of consideration for the flow table occupancy as decision tree models increase in complexity; (ii) Adaptability, e.g., unsupervised models fail to adjust themselves when exposed to real-world repetitive traffic; (iii) Robustness, e.g., the vulnerability introduced by fixed inference points at the flow level, which are prone to bypass attacks.

To overcome these challenges mentioned above, we propose a malicious traffic detection system, SentinelX, which is deployable on hardware switches. SentinelX is characterized by the following features: first, we use multiple small flow tables to replace a cumbersome and redundant large flow table, and we develop an algorithm (i.e. TreeDivider) to split the large flow table; second, we develop an unsupervised decision tree, DualTree, which uses a dual threshold model to determine the standards for identifying traffic anomalies, to cope with real traffic environments; third, we design a fine-grained inference point selection method, introducing a random algorithm in programmable switches to support the random selection of inference points. This approach effectively counters potential evasion attacks that attackers might launch.

In summary, SentinelX's contributions are threefold:

- SentinelX introduces a novel model representation method employing multiple space-saving small flow tables to replace a complicated and redundant large flow table, addressing memory constraints. This approach marks

‡These authors contributed equally to this work.
§Corresponding author: Qing Li (liq@pcl.ac.cn), Zhuyun Qi (qizhy@pcl.ac.cn)

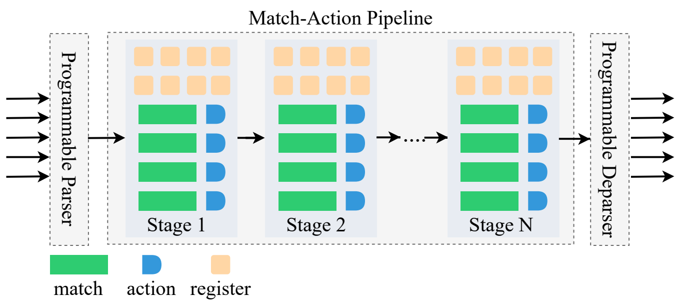


Fig. 1: Protocol-Independent Switch Architecture (PISA).

the first instance of deploying a decision tree using multiple small flow tables. Additionally, we have developed the TreeDivider algorithm to effectively split large flow tables, ensuring optimal division while maintaining the accuracy of flow table semantics.

- SentinelX proposes an unsupervised decision tree DualTree, suitable for programmable switches. DualTree employs a dual-threshold model to identify anomalous traffic, addressing challenges such as duplicate packet headers due to network retransmissions and other real-world network issues. Moreover, SentinelX includes fine-grained flow-level detection to effectively counter evasion attacks.
- We implement a prototype of SentinelX on the H3C S9830-32H-H1 switch and conduct extensive testing. The experiments demonstrate that SentinelX achieves 99.99% of the maximum throughput of the switch ports and nanosecond-level latency, which is only 1.38 times the latency of basic L3 forwarding. After using the TreeDivider algorithm to partition twice, it achieves an average space saving of 63.88%. For bypass attacks, fine-grained inference point selection improves the detection rate by 30.03%.

II. Background and Related Work

A. Programmable Switches

The Protocol-Independent Switch Architecture (PISA) is proposed to address the challenges faced by traditional network switches in supporting new protocols and services. Traditional switches are often tightly tied to specific network protocols, requiring hardware changes or upgrades when new protocols and services are introduced. Such processes can be costly and time-consuming. PISA aims to provide a protocol-independent switch architecture, enabling greater flexibility in adapting to new network protocols and services without extensive hardware modifications. With PISA, switches can be adapted to new protocols and services without modifying their hardware thanks to its programmable data plane architecture. This flexibility is crucial in the ever-evolving landscape of network environments and application demands.

As illustrated in Fig.1, packets first enter a parser for header parsing, then pass through multiple match/action stages for packet operations, and finally reach the deparser for packet serialization. The parser, the match/action units, and the deparser can all be programmed to implement desired protocols. The programmable switch pipeline is sufficiently flexible, allowing direct programming through domain-specific languages like P4. The Match-Action Pipeline supports exact matches, ternary matches, and longest prefix matches. Each match is associated with an action, where specific computations and storage modifications can be executed. Interdependent actions need to be placed in different stages. The header and metadata instances utilize stateless storage, reinitializing with the arrival of new packets. PISA also provides stateful and persistent storage options, such as counters, meters, and registers. Although programmable switches offer many advantages, they also have some limitations.

- **Memory Limitation.** Each stage is equipped with two high-speed types of memory. The first is TCAM, a content-addressable memory well-suited for rapid table lookups. TCAM stores entries that involve matching types such as ternary, longest prefix matching (LPM), and range matching. It's worth noting that not all programmable switches are suitable for range matching [9]. The second type is SRAM, utilized for storing entries that require exact matching in tables and status registers. SRAM has a capacity of approximately 100MB, while TCAM is significantly smaller than SRAM [8].
- **Width Limitation.** Width constraints are present in the high-speed lookup hardware TCAM of programmable switches. TCAM has width limitations, typically allowing input widths of only 36, 72, 144, or 288 bits [13], thus often limiting the capability for high-dimensional (multi-width) data matching.
- **Operation Restrictions.** Programmable switches impose operational constraints that exclude complex instructions such as division and floating-point operations [8]. Packet processing in the Match-Action Pipeline is limited to basic instructions, such as integer addition and bit shifting. These permitted instructions have also been limited in terms of their number within specific operations.

B. Related Work

TABLE I: Comparison with prior art in programmable switches.

Prior Works	Line-Speed	Unsupervised Model	Flow-Level	High Dimensionality
Mousika [8]	✓	✗	✗	✗
SwitchTree [11]	✓	✗	✓	✗
Flowrest [12]	✓	✗	✓	✗
Netbeacon [9]	✓	✗	✓	✗
HorusEye [7]	✓	✓	✓	✗
SentinelX	✓	✓	✓	✓

Implementing a malicious traffic detection system on programmable switches is a promising endeavor. TABLE I provides an overview description of existing solutions.

1) Malicious Traffic Detection: Although Mousika [8] and SwitchTree [11] can achieve line-rate speeds, they cannot deploy high-dimensional models due to memory constraints. In contrast, Flowrest [12], Netbeacon [9], and HorusEye [7] improve model accuracy by utilizing flow-level features. However, their disadvantage lies in performing inference only at fixed points, where attackers can easily identify and circumvent, posing significant security risks. HorusEys uses an unsupervised model for malicious traffic detection, but it suffers from limited accuracy due to the lack of consideration for packet retransmission. Our method, DualTree, employs a dual-threshold approach, performing a secondary assessment for decisions made at decision tree leaf nodes to reduce false positives. Experiments show that our approach improves the accuracy of detecting anomalous traffic by approximately 30.24% compared to traditional unsupervised decision trees [7], [14]–[19].

2) Processing of Flow-Level Features: Mousika [8] only detects at the packet level, thereby missing the opportunity to enhance accuracy through flow-level features. SwitchTree [11] and HorusEye [7] use non-statistical flow-level features for detection, but overlooks the role of statistical flow-level characteristics. Flowrest [12] and NetBeacon [9] can only support fixed inference points at powers of two due to the shift operation in programmable switches. Because these inference points are fixed, attackers can easily bypass the system to carry out attacks. Our flow-level detection method specifically studies attackers' tactics against statistical flow features and has made corresponding countermeasures. Compared to approaches that ignore such actions, our method has improved detection accuracy by about 30.03%.

3) Model Deployment in Programmable Switches: SwitchTree [11] leverages if-else statements in programmable switches to represent decision tree branches. Deploying if-else statements requires occupying different stages in the switch, which limits deployments to low-dimensional, simple decision trees. Mousika [8], Netbeacon [9], and Flowrest [12] encode decision trees into ternary rules, which are installed in the programmable switch's TCAM for matching. The width of the ternary rules generated by these methods matches the number of branch nodes in the decision tree. However, since the TCAM width is limited, models cannot be deployed if the ternary rule width exceeds TCAM's capacity, preventing the deployment of high-dimensional decision trees. HorusEye [7] uses range matching that integrates branch node features into final range matching rules using a Cartesian product. However, this approach has two significant flaws: (1) Not all programmable switches support range matching; and (2) An exponential increase in complexity occurs when adding branch node features due to the use of a Cartesian product for integration, making deployment less feasible on switches with limited memory. Our method encodes decision trees into ternary matching rules and employs multiple small flow tables instead of large ones. This solves the limited TCAM width issue and significantly reduces the switch space occupied by eliminating redundancy in large flow tables.

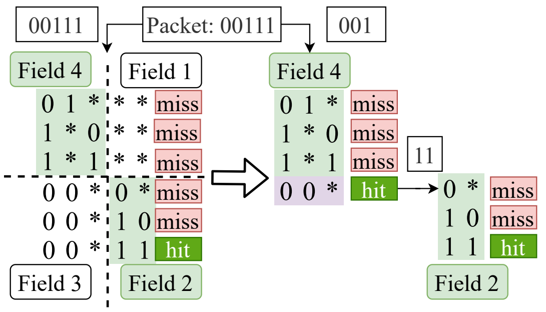


Fig. 2: An example of cutting the match field of a flow table.

III. Motivation and System Overview

A. Motivation Example

When decision trees generate flow tables composed of ternary rules through encoding, we find that there is no overlap between any two rules, nor does complex rule dependency occur. We begin with this as a basis for our work. With this characteristic, we do not need to consider complex dependencies between table entries [20], [21], allowing us to streamline the flow table. By removing complex and redundant information from the flow table and reorganizing it, we use multiple simplified small flow tables to replace the previous large flow tables. Below is a simple example to illustrate our idea.

As shown in Fig.2, on the left is a flow table generated by a decision tree through encoding, which was a large flow table containing redundant information before transformation, with a width of 5 bits. It is evident that Field 1 and Field 3 are the redundant information in this table because even without the part of Field 1, the packet can complete the match of the information in Field 4; similarly, even without the part of Field 3, the match can be completed through the information in Field 2. On the right are two refined, fine-grained flow tables after the transformation. We added an entry from Field 3 to the Field 4 part to serve as a pointer pointing to the Field 2 part. In programmable switches, packet matching through table entries follows a top-down priority order. When a match occurs, the packet first matches in the Field 4 part, then matches in the Field 2 part, resulting in a match outcome consistent with that before the transformation. For example, in the diagram, we have a packet with a matching domain of "00111", which, in the order from top to bottom, hits the entry "00*11" in the table before the transformation. After the transformation, it first passes through the Field 4 part, hitting the last entry, namely the one pointing to the Field 2 part, thus requiring continued matching; otherwise, it could directly terminate. In the Field 2 part, it hits the "11" entry, matching the same entry as in the large table before the transformation, thereby proving that our large flow table before the transformation and the small flow table after the transformation are semantically consistent.

Through such a simple splitting and transformation, we have reduced the maximum width of the overall flow table from 5 bits to 3 bits. This reduction in maximum width helps us break through the maximum match width limitation of TCAMs in programmable switches, allowing us to deploy models of higher complexity and dimensionality. As for memory usage, the flow table space has been reduced from 30 bits to 18 bits, resulting in a compression rate of 40% while maintaining semantic consistency. This is undoubtedly beneficial for the tight memory space in programmable switches.

Overall, we divide large flow tables into smaller ones through a multi-level structure, enhancing match and lookup efficiency. Thanks to the high performance of the hardware, the slight increase in latency caused by additional searches (in nanoseconds) can be negligible.

B. System Overview

As shown in the Fig. 3, SentinelX is primarily divided into two parts: the control plane and the data plane. In the control plane, we have proposed a novel unsupervised decision tree algorithm called DualTree, which is capable of detecting 0-day attacks while better adapting to the hardware characteristics of programmable switches. To address the challenges of inseparable data points and potential duplications, we have introduced a dual-threshold method to enhance detection accuracy. To ensure the model's lightweight nature, we designed a decision tree-oriented splitting algorithm, TreeDivider, which creates subtrees consisting of non-leaf and leaf nodes. The non-leaf nodes are defined by the splitting features and integer values, whereas the leaf nodes include three types: (1) benign traffic; (2) malicious traffic; and (3) linked subtrees.

At the data plane, we proposed, for the first time, the use of multi-level flow tables in programmable switches to represent the model. By deploying the model split by TreeDivider, we can reduce the memory usage of the switches and deploy larger-scale models. In terms of flow-level processing, packets first need to undergo a blacklist scan. If identified as malicious, they are dropped; otherwise, the features in the register are updated. Next, the inference point is selected. If it isn't an inference point, it indicates a benign flow that should be forwarded. Then, the feature encoding phase begins, and the flow table of the model is queried to produce a result. Subsequently, the blacklist is updated, allowing for the effective detection of short flows and the prevention of bypass attacks.

IV. Packet Processing

The characteristics of a data packet originate from two aspects: (i) packet-level features and (ii) flow-level features. The flow-level features are further divided into two parts. One part includes flow-level features that every packet has, such as the total packet length of the flow to which the packet belongs, and the total number of packets in that flow. The other part consists of statistical flow features, such as the average packet length and average number of packets in the flow. Previous works in Flowrest [12] and Netbeacon [9] calculate the statistical features of the flow using bit-shifting operations supported by programmable switches. However, such calculations are limited because they can only be performed at powers of two, thus these studies set their inference points at powers of two, which makes it impossible to detect attacks on packets that are not powers of 2. Our flow-level scheme aims to solve the above problems, enhancing the robustness of detection as well as improving the accuracy of detection.

Algorithm 1: Data Level Packet Processing

```
Input: $PIn$: PacketIn
1  HashKey = Hash(PIn)                                  // get HashKey
2  if PIn in blackList then
3      PIn.drop()                                       // malicious flow
4  end if
5  UpdateFeatures(HashKey, PIn)                         // update features
6  if $PIn.id \leq 8$ or genRandom(PIn.timestamp) match
     tableRandom then
7      PIn.infer = 1                                    // need to infer
8  end if
9  if PIn.infer == 1 then
10     PIn.result = infer(PIn)                          // infer packet result
11     if PIn.result == 1 then
12         PIn.forward()                                // benign flow
13     else
14         PIn.drop()                                   // malicious flow
15         UpdateBlackList(PIn)                         // update blackList
16     end if
17 else
18     PIn.forward()                                    // no need to infer
19 end if
```

As shown in Algorithm 1, packets are first hashed upon arrival to compute a 32-bit HashKey (Line 1). We have discussed our hash collisions in Section VI. Then, it is determined whether the packet's stream is on the blacklist; if identified as a malicious stream, it is directly dropped (Line 2-4). The features of packets that are not from a malicious stream are updated (Line 5). Next, it is assessed whether the packet is one of the first eight, or a random selection is made for packets beyond the first eight to determine the inference points (Line 6). If it is not an inference point, the packet is forwarded (Line 17-19) to avoid data delays. This method prevents attackers from bypassing flow-level inference attacks. Then, inference is carried out on the packets at the inference points (Line 9-10). Following this, the results of the packet inference are used to decide the destination of the packets and whether to update the blacklist (Line 11-16).

V. Model Building

It is difficult to separate certain data points when dealing with network traffic datasets using unsupervised decision trees that use integers for splitting. To address this, we designed DualTree in Section V-A, which uses dual thresholds to detect network traffic datasets, effectively enhancing detection efficiency. Deploying unsupervised decision trees in programmable switches presents the following problems:

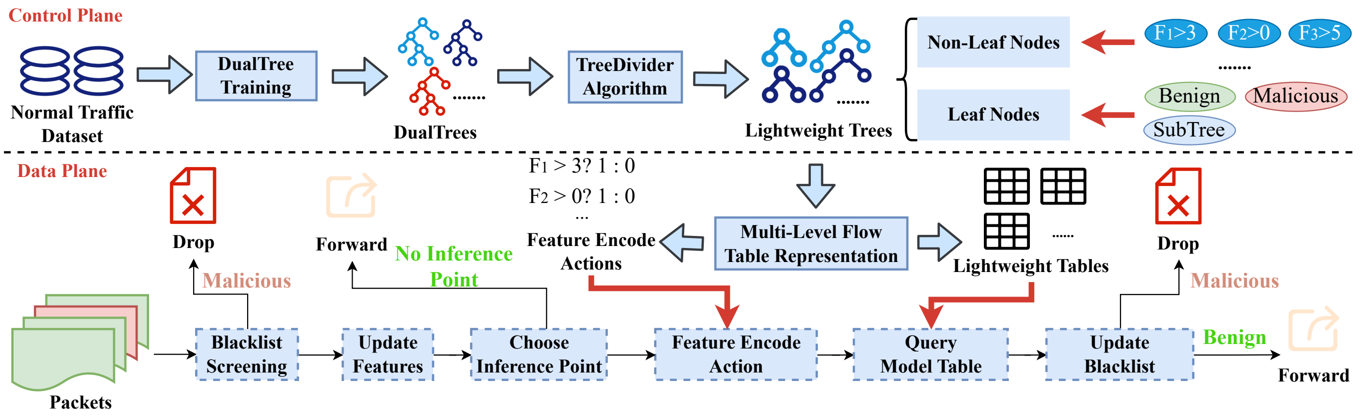


Fig. 3: The architecture of SentinelX.

TABLE II: Duplicate rate of data points in different types of dataset.

Dataset	Duplicate Rate	Dataset	Duplicate Rate
NSL-KDD	0.8239	CICDS 2017	0.1171
Aposemat IoT-23	0.9990	CIC IoT 2023	0.5396
Encrypted 2021	0.1949	Encrypted 2022	0.1305

(1) Due to width limitations in the TCAM of programmable switches, flow tables with excessively wide match fields cannot be supported (for example, the TCAM of Tofino 1 can only support up to 524-bit match width); (2) Programmable switches have memory limitations (e.g., TCAM in Tofino 1 has only 6.2 MB [9]) and computational limitations (unable to support floating-point numbers and division operations, etc.). To resolve these issues, we propose a multi-level flow table to represent the decision tree in Section V-B, and introduce a specific method for splitting, TreeDivider, in Section V-C.

A. DualTree

In traditional unsupervised anomaly detection, data points are split as much as possible to distinguish between normal and abnormal traffic. Within the maximum allowed tree height, each leaf node must contain at most one data point. However, deploying unsupervised decision trees on switches requires integer split values, as programmable switches do not support floating-point operations. This introduces a problem: during training, integer split values cannot thoroughly partition data points, resulting in some leaf nodes containing more than one data point. This situation arises due to the following three reasons: (1) Flow-level features might be floating-point numbers. (2) The dataset may contain duplicate data points due to reasons such as packet retransmission. (3) After feature selection, originally distinct data points might become identical due to the reduction in features.

To verify the hypothesis mentioned above, we conducted experiments on network traffic datasets, IoT traffic datasets, and encrypted traffic datasets, as illustrated in the TABLE. II. We converted all floating-point features to integer features to examine the prevalence of duplicate data points within these datasets. The results revealed a significant presence of duplicate data points across all types of traffic. IoT traffic exhibited the highest proportion of duplicates, with 77% of data points showing repetitions.

Algorithm 2: DualTree

```
   Input: $t_1$: Threshold 1, $t_2$:Threshold 2, $T$:Trained Tree
   Result: $T$:DualTree
1  for node in T.LeafNodes() do
2      node.result = 0                         // init as normal node
3      if $node.height/allHeight \leq t_1$ then
4          node.result = 1                     // $t_1$: init malicious node
5          if $node.number/allNumber > t_2$ then
6              node.result = 0                 // $t_2$: rejudge normal node
7          end if
8      end if
9  end for
10 return $T$
```

To address the issue of difficult data point partitioning during training, we designed DualTree, which uses dual thresholds to improve the original threshold problem [7], [14]. As shown in Algorithm 2, it starts by traversing all leaf nodes (Line 1) and initializing each node as a normal node (Line 2). Then, it checks Threshold 1: if the height of the current leaf node is less than t1 relative to the total height, the node is considered abnormal. This is also how traditional methods [7], [14] determine abnormal nodes. Otherwise, a second judgment is made: if the number of data points assigned to the leaf node during training exceeds t2 of the total data points (Line 5), the node is considered normal (Line 6). This is because the node has a larger number of data points assigned to it, rather than the single data point envisioned by the decision tree, making it likely that normal data points are difficult to partition due to integer split values. Generally, abnormal data points are discrete, so leaf nodes with abnormal data points are more likely to contain a single data point.

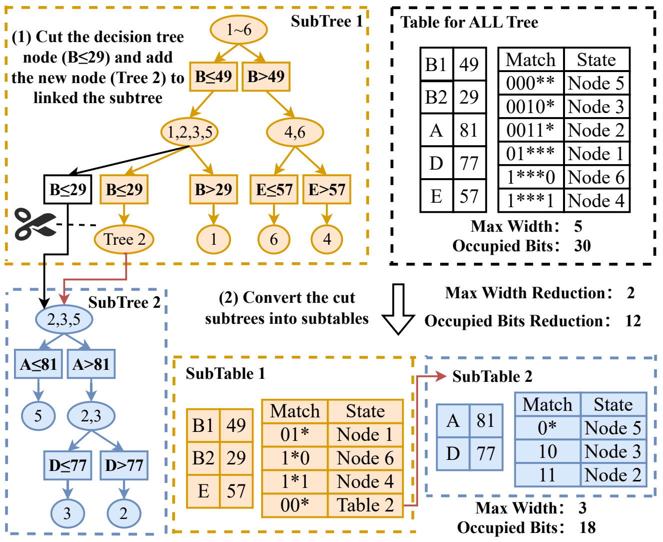


Fig. 4: Model representation method of SentinelX.

B. *Multi-Level Flow Table Representation*

Our multi-level flow table representation method differs from the current mainstream feature encoding methods [8], [9], [12]. SentinelX uses a multilevel flow table format to represent a decision tree, allowing more complex models to be deployed in programmable switches at a lower cost.

As shown in the Fig. 4, the upper right is the flow table generated using the entire decision tree. When a packet enters the switch, it first goes through feature encoding that converts the packet into a binary string format. This binary string then enters the flow table for matching. For example, a packet with attributes A=90, B=20, C=30, D=66, E=60 undergoes feature encoding starting with B1, where B=20 is less than 49, hence the first bit encoded is "0". Next, comparing B2, B=20 is less than 29, resulting in the second bit encoded as "0", and so on, until comparing E where E=60 is greater than 57, making the final bit encoded as "1". The final binary string encoded is "00101". In standard feature encoding, this results in Node 3; the flow table occupies a width of 5 and requires 30 bits of space.

The lower right of Fig. 4 shows two linked flow tables for the multi-level flow table representation scheme. We split the area above the non-leaf node (2, 3, 5), dividing it into SubTree 1 and SubTree 2, and then converting SubTree 1 and SubTree 2 into SubTable 1 and SubTable 2, respectively. The two resulting flow tables have a maximum width of 3 and occupy a total space of 18 bits. From this example, it is clear that the multilevel flow table model representation method not only reduces the space required but also shortens the maximum width occupied by the flow table.

To ensure semantic accuracy, we split from the original decision tree rather than from an existing flow table. By converting the flow tables from the decision tree, there is no need to consider rule dependency between any two flow tables, as there is no overlap in the ranges between any two rules, and thus no need for priority information. Flow tables converted from a decision subtree do not have complex dependencies since it is still a decision tree. In contrast, splitting an existing flow table would create complex dependencies, thereby losing the semantic correctness.

In the example on the right side of Fig. 4, if the same packet {A=90, B=20, C=30, D=66, E=60} enters, it would first be encoded as "001" and then go into SubTable 1 for matching. In SubTable 1, it matches the last item, which necessitates a match in SubTable 2. Before entering SubTable 2, it would be encoded as "10" thereby matching Node 3, consistent with the result on the left side. This demonstrates that even though the representation method has changed, the accuracy and the outcome remain the same.

Algorithm 3: TreeDivider

```
Input: $T$: Tree, $K$: Number of subtrees after splitting
Result: $newTrees$: Subtrees after splitting
1  Function cutTree(G, subNonLeaf):
2      subRoot = findBestSubRoot(G, subNonLeaf)
3      parent = G.predecessors(subRoot)       // find parent
4      newTree = G.subTree(subRoot)           // get subtree
5      G.removeEdge(parent, subRoot)          // remove edge
6      G.addLeaf(parent)       // add leaf node after parent
7      return G, newTree
8  End Function
9  treeGraph = T.getGraph()                   // get graph
10 nonLeaf = treeGraph.getNonLeafNumber()
11 subNonLeaf = nonLeaf / K         // get subtree non-leaf
12 newTrees = []                              // init result
13 for $t$ = 1 to $K$-1 do
14     treeGraph, newSubtree = cutTree(treeGraph,
        subNonLeaf)
15     newTrees.add(newSubtree)               // Add subtree
16 end for
17 newTrees.add(treeGraph)               // Add root subtree
```

C. *TreeDivider*

If the original uncut decision tree T has N_0 non-leaf nodes, then the number of leaf nodes is $(N_0 + 1)$, and the size of the flow table generated is $S_0 = N_0(N_0 + 1)$. If the original decision tree is cut $(k - 1)$ times, resulting in k decision subtrees, and the number of non-leaf nodes for these k subtrees are: N_1, N_2, ..., N_k. Since the total number of non-leaf nodes remains unchanged through the cutting process, we have:

$$N_1 + N_2 + ... + N_k = N_0 \tag{1}$$

The size of the flow tables occupied by these k decision subtrees are respectively:

$$S_1 = N_1(N_1 + 1), ..., S_k = N_k(N_k + 1) \tag{2}$$

Then,

$$S_1 + S_2 + \cdots + S_k = N_1^2 + N_2^2 + \cdots + N_k^2 + N_0$$
$$\geq \frac{N_0^2}{k} + N_0 \tag{3}$$

In Equation 3, the equality holds if and only if $N_1 = N_2 = \ldots = N_k = N_0/k$. Hence, to minimize the volume of flow tables that are occupied after the cuts, all non-leaf nodes should be close to N_0/k. Therefore, we only need to cut N_0/k subtrees of non-leaf nodes on the original decision tree each time.

Algorithm 3, named TreeDivider, takes T and K as inputs, representing the decision tree to be divided and the desired number of subtrees after division, respectively. The output is newTrees, which includes all the subtrees resulting from the division. Lines 1-8 describe the function for performing the division and returning the results. Lines 9-17 contain the main function that manages the input division and the output. Line 9 converts the input decision tree into a graph form for manipulation, Line 10 calculates the number of non-leaf nodes in the current tree, and Line 11 computes the number of non-leaf nodes for each subtree. Line 12 initializes the final output, and Lines 13-16 perform $K-1$ divisions to form K subtrees. For example, in Fig. 4, with $K = 2$, the computed optimal number of non-leaf nodes for each subtree is 2. In each division, Line 1 inputs the current graph G to be divided and the number of non-leaf nodes required for the subtree being separated. Line 2 finds the optimal node for the current division, Line 3 identifies the predecessor of the optimal split node, Line 4 locates the subtree of the current optimal split node, Lines 5-6 perform the division, and Line 7 returns the results of the division.

VI. Evaluation

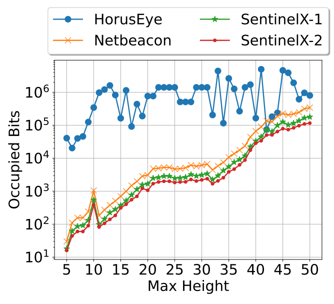
A. Experimental Settings

At the control level, we have implemented the unsupervised decision tree algorithm DualTree and the decision tree splitting algorithm TreeDivider using Python. Our experiments were conducted on a Supermicro SYS-7049GP-TRT server, which is equipped with two Intel Xeon Gold 6230R CPUs.

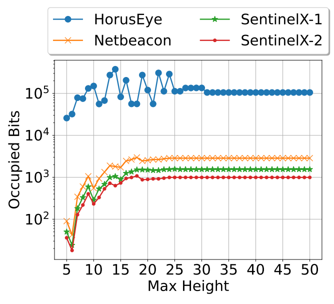
In terms of data processing, we employed the P4 language to facilitate flow-level feature processing and deployed a trained SentinelX model on the H3C S9830-32H-H1 switch.

B. Task and Dataset Setup

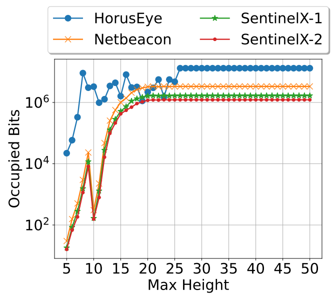
We configured three distinct detection tasks in SentinelX: Network Malicious Traffic Detection(NSL-KDD [22], [23] and CICIDS 2017 [24]), IoT Malicious Traffic Detection(Aposemat IoT-23 [25] and CIC IoT 2023 [26]), and Encrypted Malicious Traffic Detection(Encrypted 2021 [27] and Encrypted 2022 [28]). This configuration aimed to conduct a comprehensive performance evaluation of SentinelX. Each of these detection tasks incorporated two widely utilized open-source datasets.



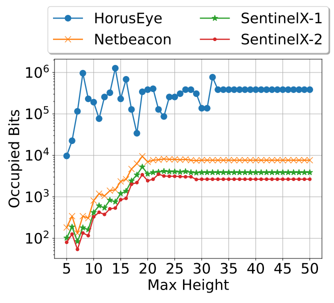
(a) NSL-KDD



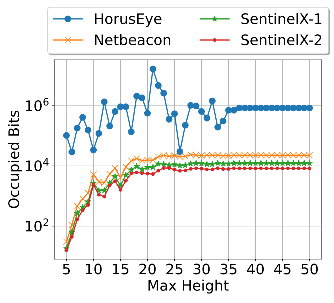
(b) CICDS 2017



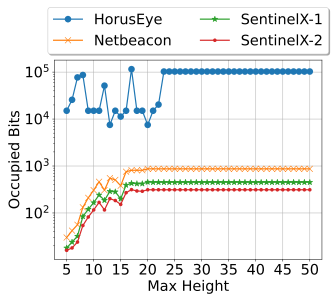
(c) Aposemat IoT-23



(d) CIC IoT 2023



(e) Encrypted 2021



(f) Encrypted 2022

Fig. 5: Comparison between HorusEye, Netbeacon, SentinelX-1 and SentinelX-2 in terms of lightweight.

C. Baseline Setup

In the lightweight evaluation of SentinelX, our SentinelX utilizes a multi-level flow table deployment model to evaluate two scenarios: (1) SentinelX-1: the scheme using TreeDivider to split once; (2) SentinelX-2: the scheme using TreeDivider to split twice. Our benchmarks are the current state-of-the-art (SOTA) work in feature encoding, Netbeacon [9], and the SOTA work in range encoding, HorusEye [7], for comparison.

In zero-day attack detection, we choose HorusEye as our benchmark for evaluation because HorusEye [7] represents the state-of-the-art (SOTA) work in deploying unsupervised learning models in programmable switches. Other works in programmable switches are primarily based on supervised learning, which cannot detect zero-day attack traffic.

In the hardware evaluation of SentinelX, we demonstrate that it can achieve high throughput and low latency on programmable switches, compared to programs that only support basic L3 forwarding. Additionally, we conduct a comprehensive evaluation of SentinelX's performance on hardware, including resource utilization, and packet loss rate.

TABLE III: Comparison of average bit usage of HorusEye, Netbeacon, SentinelX-1 and SentinelX-2.

Dataset	HorusEye	Netbeacon	SentinelX-1	SentinelX-2
[22], [23]	1103885.83	51560.74	27132.57	18764.91
[24]	124003.65	2364.57	1279.74	833.09
[25]	8562641.17	2480617.30	1245249.00	907113.65
[26]	342661.57	5895.61	3011.61	2126.70
[27]	1151987.65	16654.39	8927.52	6046.39
[28]	74349.57	720.52	375.13	260.26

D. *Task Evaluation*

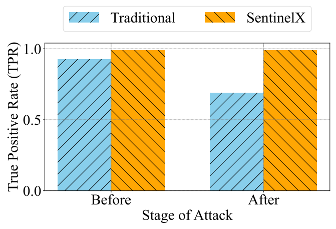
We conducted extensive evaluations of SentinelX to prove that: (i) the model representation method at the data level in SentinelX is efficient and meets the requirements for lightweight deployment; (ii) SentinelX can detect zero-day attacks and achieves high performance in detecting malicious traffic; (iii) SentinelX effectively utilizes the high performance of the deployed hardware switches, including high throughput, low latency, and low packet loss rate, among others.

1) Lightweight Performance: Fig. 5 compares the space occupied by models generated by HorusEye, Netbeacon, SentinelX-1, and SentinelX-2. The x-axis represents the maximum height of the decision trees, and the y-axis shows the space occupied by the decision trees, using a logarithmic scale. It is evident that, across almost all tasks, HorusEye occupies the most space, while SentinelX-2 occupies the least. In HorusEye, deployment uses range rules, and rule generation is based on Cartesian products, so the space it occupies grows explosively when there are many features. Netbeacon deploys using feature encoding, which does not experience explosive growth in table size with the increase in features, but it is still slightly inferior to SentinelX, as SentinelX's multi-level flow table deployment method eliminates the redundancy in feature encoding, thereby reducing the space occupied.

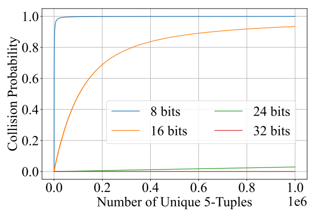
TABLE III presents a comparison of the bit occupancy numbers for model deployment across various datasets using four different methods. Methods in the SentinelX series show a clear advantage. For example, SentinelX-1, which splits only once, has a bit space reduction of 47.71% on average compared to Netbeacon, while SentinelX-2, which splits twice, shows an average reduction of 63.88%. The more splits SentinelX performs, the greater the reduction in bit space. However, SentinelX cannot split indefinitely due to the stage limitations of programmable switches. Tables that are connected in nature must be placed in consecutive stages, meaning the longest chain of tables in a multi-level flow table matching method cannot exceed the number of stages in the switch.

2) Zero-Day Attack Detection:

a) Unlabeled Attack: In this evaluation, the training set contains only normal traffic, while the test set has a normal to abnormal traffic ratio of 1:1. In HorusEye, the threshold setting method for iForest is set to "auto". Generally, SentinelX selects thresholds based on normal traffic in the training set. In the experiments, SentinelX's Threshold 1 is set between 0.1 and 0.2, and Threshold 2 is set between 0 and 0.1.



(a) Comparison between Traditional and SentinelX in dealing with bypass attack.



(b) Comparison of 8 bits, 16 bits, 24 bits and 32 bits in hashing store.

Fig. 6: Bypass Attack and Hash Collision

As can be seen from the TABLE IV, SentinelX outperforms HorusEye with higher Accuracy and TPR, and a lower FPR. The average Accuracy increased by 32.29% compared to HorusEye, and the FPR decreased by 28.68%. Due to the use of integer split values in programmable switches, it is difficult to separate some data points in the network traffic datasets, resulting in poor model performance. Our DualTree effectively addresses this issue by employing a dual-threshold mode. Unlike traditional unsupervised decision trees, our system re-evaluates the leaves that cannot be finely divided to enhance detection accuracy.

b) Bypass Attack and Hash Collision: To validate the appropriateness of our selected inference points, we compare the inference methods of SentinelX and traditional approaches in the context of evasion attacks, utilizing the Aposemat IOT-23 dataset. We assume that attackers are aware that programmable switches are more likely to perform inference at the second power of the data packets in a flow. Therefore, they insert normal data packets at the second power points of the anomalous flow to masquerade as normal traffic and achieve their attacking purpose. We use the same model for inference, comparing the differences in anomalous traffic before and after the attack under the traditional inference points and SentinelX's inference points.

Fig. 6(a) compares SentinelX with the traditional model. Since SentinelX uses the same model, it doesn't offer much advantage before bypass attacks. However, at the inference points, SentinelX provides a more thorough detection of short flows by checking the first eight packets of the flow in detail. As a result, SentinelX achieves about 6.35% higher detection rate in identifying abnormal flows compared to traditional methods. After the onset of bypass attacks, SentinelX's detection rate in abnormal flows is 30.03% higher than that of the traditional method. Since traditional inference points are rigid, they may not be able to adapt to slight changes in attacker tactics. SentinelX, incorporating pseudorandom inference points, maintains nearly unchanged accuracy in the face of such bypass attacks.

Below, we have discussed hash collisions and compared the

TABLE IV: Performance comparison between HorusEye and SentinelX.

Dataset	Attack Type	HorusEye			SentinelX		
		Accuracy	TPR	FPR	Accuracy	TPR	FPR
	Ipsweep	0.922	0.956	0.111	**0.956** (+3.69%)	**0.978** (+2.3%)	**0.067** (-39.63%)
	Back	0.950	1.000	0.100	**1.000** (+5.26%)	**1.000** —	**0.000** (-100%)
	Nmap	0.750	0.583	0.083	**0.958** (+27.73%)	**0.917** (+57.3%)	**0.000** (-100%)
	Pod	1.000	1.000	0.000	**1.000** —	**1.000** —	**0.000** —
	Teardrop	0.583	0.333	0.167	**0.833** (+42.88%)	**0.667** (+100.3%)	**0.000** (-100%)
	Warezclient	0.750	0.750	0.250	**1.000** (+33.33%)	**1.000** (+33.33%)	**0.000** (-100%)
	Slowhttptest	1.000	1.000	0.000	**1.000** —	**1.000** —	**0.000** —
[22], [23]	GoldenEye	0.500	0.000	0.000	**1.000** (+100%)	**1.000** ($+\infty$)	**0.000** —
[24]–[26]	FileDownload	0.705	0.591	0.182	**0.750** (+6.38%)	**0.682** (+15.4%)	**0.182** —
	DDoS-ACK	0.500	0.143	0.143	**0.857** (+71.4%)	**0.857** (+499.31%)	**0.143** —
	DNS-Spoofing	0.429	0.000	0.143	**0.643** (+49.89%)	**0.429** ($+\infty$)	**0.143** —
	ArpSpoofing	0.750	0.667	0.167	**0.833** (+11.07%)	**0.833** (+24.89%)	**0.167** —
	DDoS-UDP	0.438	0.000	0.125	**0.938** (+114.16%)	**1.000** ($+\infty$)	**0.125** —
	HostDiscovery	0.500	0.000	0.000	**0.667** (+33.4%)	**0.333** ($+\infty$)	**0.000** —
	SlowLoris	1.000	1.000	0.000	**1.000** —	**1.000** —	**0.000** —
[27]	Encrypted 2021	0.651	0.417	0.114	**0.717** (+10.14%)	**0.544** (+30.46%)	**0.110** (-3.51%)
[28]	Encrypted 2022	0.522	0.162	0.119	**0.729** (+39.66%)	**0.524** (+223.46%)	**0.066** (-44.54%)

TABLE V: Performance of SentinelX on hardware switches.

Model	Stages	SRAM	TCAM	Forward Delay(ns)
L3 Basic Forward	2	5%	0%	751
NSL-KDD	11	8.52%	0.76%	1032
CICDS 2017	11	7.61%	0.76%	1032
Aposemat IoT-23	11	3.98%	0.76%	1030
CIC IoT 2023	11	11.02%	1.14%	1035
Encrypted 2021	11	10.68%	1.14%	1038
Encrypted 2022	11	10.57%	1.14%	1036

reasoning points of our stream with the fixed reasoning points of previous work on the dataset. In terms of hash collisions, we use a 32-bit method to save the HashKey. As shown in the Fig. 6(b), the probability of hash collisions generally increases with the number of streams. At the same time, comparing the storage of different bit lengths, the probability of hash collisions for five-tuples significantly decreases as the number of storage bits increases. Using 32 bits for storage, there is almost no chance of a hash collision occurring.

3) Hardware Performance: We evaluated the SentinelX models, each trained on six distinct datasets, by deploying them on hardware switches. The evaluation focused on three key metrics: (1) Throughput, which refers to the packet reception and transmission rates of the switch after loading the program; (2) Resource Utilization, measured by the usage rates of SRAM and TCAM on the switch; and (3) Latency, denoted by the time taken by the switch to process each packet.

TABLE V shows the results of our tests conducted on a hardware switch after installing various models. We used the SPIRENT N11U traffic generator for high-speed traffic simulation, sending approximately 10,775,862,068 packets without experiencing any packet loss, resulting in a packet loss rate of 0%. Regarding throughput, we tested six SentinelX models on the switch's 100Gb port. Each model achieved a stable throughput of 99,999,999,991 bps, which is close to the L3 Basic Forward's throughput of 99,999,997,440 bps, representing 99.99% of the port's maximum throughput. In terms of resource usage, with each model including a flow-level feature awareness module (occupying 0-4 stages), the average SRAM usage across the six models was only 8.73%, the average TCAM usage was only 0.95%, and the average number of stages occupied was only 11. Regarding latency, the average wait time delay for our six models was only 1,033.83 ns, just 1.38 times that of L3 Basic Forward.

VII. Conclusion

SentinelX first introduced a multi-level flow table representation method, making the deployed model more lightweight and addressing the issue of TCAM width restrictions in model deployment. It also proposed the TreeDivider algorithm to divide the model more precisely. In terms of flow-level processing, SentinelX designed a more optimal inference point method to better detect short flows and counter bypass attacks. To better address the challenges of dividing data points under the constraints of programmable switches, SentinelX designed a dual-threshold unsupervised decision tree. Finally, we implemented the SentinelX model on hardware switches and evaluated its performance across a broad range of tasks. Future work will further optimize SentinelX's algorithms to adapt to more complex network environments and explore its performance for different types of network attacks. Additionally, research will be conducted on how to apply and scale SentinelX in larger networks to validate its effectiveness and stability in real-world environments.

Acknowledgment

This work is supported by the Major Key Project of PCL under grant No. PCL2023A06, the National Key Research and Development Program of China under grant No. 2022YFB3105000, and the Shenzhen Key Lab of Software Defined Networking under grant No. ZDSYS20140509172959989.

References

[1] Mamoona Humayun, Bushra Hamid, NZ Jhanjhi, G Suseendran, and MN Talib. 5g network security issues, challenges, opportunities and future directions: A survey. In *Journal of Physics: Conference Series*, volume 1979, page 012037. IOP Publishing, 2021.

[2] Yushan Siriwardhana, Pawani Porambage, Madhusanka Liyanage, and Mika Ylianttila. Ai and 6g security: Opportunities and challenges. In *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pages 616–621. IEEE, 2021.

[3] Wan Haslina Hassan et al. Current research on internet of things (iot) security: A survey. *Computer networks*, 148:283–294, 2019.

[4] Razvan Bocu, Maksim Iavich, and Sabin Tabirca. A real-time intrusion detection system for software defined 5g networks. In *International Conference on Advanced Information Networking and Applications*, pages 436–446. Springer, 2021.

[5] Chuanpu Fu, Qi Li, Meng Shen, and Ke Xu. Realtime robust malicious traffic detection via frequency domain analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3431–3446, 2021.

[6] Bo Wang, Yang Su, Mingshu Zhang, and Junke Nie. A deep hierarchical network for packet-level malicious traffic detection. *IEEE Access*, 8:201728–201740, 2020.

[7] Yutao Dong, Qing Li, Kaidong Wu, Ruoyu Li, Dan Zhao, Gareth Tyson, Junkun Peng, Yong Jiang, Shutao Xia, and Mingwei Xu. Horuseye: A realtime iot malicious traffic detection framework using programmable switches. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 571–588, 2023.

[8] Guorui Xie, Qing Li, Yutao Dong, Guanglin Duan, Yong Jiang, and Jingpu Duan. Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 1938–1947. IEEE, 2022.

[9] Guangmeng Zhou, Zhuotao Liu, Chuanpu Fu, Qi Li, and Ke Xu. An efficient design of intelligent network data plane. In *32nd USENIX Security Symposium (USENIX Security 23). Anaheim, CA: USENIX Association*, 2023.

[10] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. pforest: In-network inference with random forests. *arXiv preprint arXiv:1909.05680*, 2019.

[11] Jong-Hyouk Lee and Kamal Singh. Switchtree: in-network computing and traffic analyses with random forests. *Neural Computing and Applications*, pages 1–12, 2020.

[12] Aristide Tanyi-Jong Akem, Michele Gucciardo, and Marco Fiore. Flowrest: Practical flow-level inference in programmable switches with random forests. In *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2023.

[13] Chad R Meiners, Alex X Liu, and Eric Torng. Algorithmic approaches to redesigning tcam-based systems. In *Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 467–468, 2008.

[14] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[15] Songqiao Han, Xiyang Hu, Hailiang Huang, Minqi Jiang, and Yue Zhao. Adbench: Anomaly detection benchmark. *Advances in Neural Information Processing Systems*, 35:32142–32159, 2022.

[16] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A Zuluaga. Usad: Unsupervised anomaly detection on multivariate time series. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3395–3404, 2020.

[17] Hongzuo Xu, Guansong Pang, Yijie Wang, and Yongjun Wang. Deep isolation forest for anomaly detection. *IEEE Transactions on Knowledge and Data Engineering*, 35(12):12591–12604, 2023.

[18] Antonella Mensi and Manuele Bicego. Enhanced anomaly scores for isolation forests. *Pattern Recognition*, 120:108115, 2021.

[19] Nannan Dong, Baoquan Ren, Hongjun Li, Xudong Zhong, Xiangwu Gong, Junmei Han, Jiazheng Lv, and Jianhua Cheng. A novel anomaly score based on kernel density fluctuation factor for improving the local and clustered anomalies detection of isolation forests. *Information Sciences*, 637:118979, 2023.

[20] Ying Wan, Haoyu Song, Yang Xu, Yilun Wang, Tian Pan, Chuwen Zhang, and Bin Liu. T-cache: Dependency-free ternary rule cache for policy-based forwarding. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 536–545. IEEE, 2020.

[21] Zeyu Luan, Qing Li, Zutao Zhang, Yong Jiang, Meng Chen, Yu Wang, and Kejun Li. Awesome-cache: dependency-free rule-caching for arbitrary wildcard patterns in tcam. In *2023 IEEE 31st International Conference on Network Protocols (ICNP)*, pages 1–12. IEEE, 2023.

[22] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications*, pages 1–6. IEEE, 2009.

[23] John McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security (TISSEC)*, 3(4):262–294, 2000.

[24] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.

[25] Sebastian Garcia, Agustin Parmisano, Maria Jose Erquiaga, Veronica Valeros, and Maria Rigaki. IoT-23: A labeled dataset with malicious and benign IoT network traffic, May 2021.

[26] Euclides Carlos Pinto Neto, Sajjad Dadkhah, Raphael Ferreira, Alireza Zohourian, Rongxing Lu, and Ali A Ghorbani. Ciciot2023: A real-time dataset and benchmark for large-scale attacks in iot environment. 2023.

[27] Zihao Wang, Kar Wai Fok, and Vrizlynn Thing. Composed encrypted malicious traffic dataset for machine learning based encrypted malicious traffic analysis. *Mendeley Data*, 2, 2021.

[28] Zihao Wang and Vrizlynn Thing. Encrypted Traffic Feature Dataset for Machine Learning and Deep Learning based Encrypted Traffic Analysis, 2022.