

Flow Interaction Graph Analysis: Unknown Encrypted Malicious Traffic Detection

Chuanpu Fu¹, Qi Li², *Senior Member, IEEE*, and Ke Xu¹, *Fellow, IEEE, Member, ACM*

Abstract—Nowadays traffic on the Internet has been widely encrypted to protect its confidentiality and privacy. However, traffic encryption is always abused by attackers to conceal their malicious behaviors. Since encrypted malicious traffic is similar to benign flows, it can easily evade traditional detection. In particular, the existing encrypted traffic detection methods are supervised which rely on the prior knowledge of known attacks (e.g., labeled datasets). Detecting unknown encrypted malicious traffic, which does not require prior knowledge, is still an open problem. In this paper, we propose HyperVision, an unsupervised machine learning (ML) based malicious traffic detection system. Particularly, HyperVision is able to detect unknown patterns of encrypted malicious traffic by utilizing a graph built upon flow interaction patterns, instead of learning the features of specific known attacks. We develop an unsupervised graph learning method to detect abnormal interaction patterns by analyzing the graph features, which allows HyperVision to detect unknown attacks without requiring any labeled datasets. Moreover, we establish an information theory model to prove the effectiveness of HyperVision. We show the performance of HyperVision by real-world experiments with 140 attacks. The experimental results illustrate that HyperVision outperforms the state-of-the-art methods by 13.9% accuracy improvement. Moreover, HyperVision achieves 15.82 Mpps detection throughput with the average detection latency of 0.29s.

Index Terms—Malicious traffic detection, machine learning, graph learning.

I. INTRODUCTION

TRAFFIC encryption has been widely adopted to protect the information delivered on the Internet. Over 80% websites adopted HTTPS to prevent data breach in 2019 [1], [2]. However, the cipher-suite for such protection is double-edged. In particular, the encrypted traffic also allows attackers to conceal their malicious behaviors, e.g., malware campaigns [3], exploiting vulnerabilities [4], and data

exfiltration [5]. The ratio of encrypted malicious traffic on the Internet is growing significantly [3], [6], [7] and exceeds 70% of the entire malicious traffic [1].

However, encrypted malicious traffic detection is not well addressed due to the low-rate and diverse traffic patterns [3], [5], [8]. The traditional signature based methods that leverage deep packet inspection (DPI) are invalid under the attacks with the encrypted payloads [9]. Table I compares the existing malicious traffic detection methods. Different from plain-text malicious traffic, the encrypted traffic has similar features to benign flows and thus can evade existing ML based detection systems as well [2], [3], [7]. Particularly, existing encrypted traffic detection methods are supervised which rely on the prior knowledge of known attacks. For model training, they extract features from labeled datasets [3], [6], [7]. Thus, they are unable to detect a broad spectrum of attacks with encrypted traffic [4], [5], [8], [10], which are constructed with unknown patterns [11]. Besides, these methods are incapable of detecting both attacks constructed with and without encrypted traffic and unable to achieve generic detection because features of encrypted and non-encrypted attack traffic are significantly different [3], [7].

In a nutshell, the existing methods cannot detect encrypted malicious traffic with unknown patterns by learning traffic features of a single flow [3], [6]. However, it is still feasible to detect such attack traffic because these attacks involve multiple attack steps with different flow interactions among attackers and victims, which are distinct from benign flow interactions patterns [8], [12], [13], [14], [15]. For example, the encrypted flow interactions between spam bots and SMTP servers are significantly different from the legitimate communications [15] even if the single flow of the attack is similar to the benign one. Thus, this paper explores utilizing flow interaction patterns for malicious traffic detection.

To the end, we propose HyperVision, a realtime detection system that aims to capture encrypted malicious traffic. In particular, it can detect encrypted malicious flows with unknown patterns by identifying abnormal flow interactions, i.e., the interaction patterns that are distinct from benign ones. To achieve this, we build a graph to represent flow interaction patterns which allows HyperVision to perform detection on both encrypted and plain-text malicious traffic by learning the graph structural features. Meanwhile, by learning the graph structural features, it realizes unsupervised detection, which does not require model training with labeled datasets.

However, it is challenging to build the graph for realtime detection. We cannot simply use IP addresses as vertices and flows as edges to construct the graph because the resulting dense graph cannot be analyzed by graph learning methods. To address this issue, we utilize two strategies to record different

Manuscript received 24 April 2023; revised 17 September 2023; accepted 17 February 2024; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Z. Lin. Date of publication 19 March 2024; date of current version 20 August 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB3102301; in part by Beijing Outstanding Young Scientist Program under Grant BJJWZYJH01201910003011; in part by China National Funds for Distinguished Young Scientists under Grant 61825204; and in part by the National Natural Science Foundation of China under Grant 62132011, Grant 61932016, and Grant 62221003. (*Corresponding author: Ke Xu.*)

Chuanpu Fu and Ke Xu are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: fuchuanpu@gmail.com; xuke@tsinghua.edu.cn).

Qi Li is with the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China (e-mail: qli01@tsinghua.edu.cn).

This article has supplementary downloadable material available at <https://doi.org/10.1109/TNET.2024.3370851>, provided by the authors.

Digital Object Identifier 10.1109/TNET.2024.3370851

TABLE I
THE COMPARISON WITH THE EXISTING METHODS OF MALICIOUS TRAFFIC DETECTION

Data Source Categories	Data Sources	Typical Methods	Data for Detection		Design Goals			Detection Performance	
			Unlabeled Datasets	Multi-Flow Features	Generic Detection	Realtime Detection	Unknown Attacks	Low Latency	High Throughput
Encrypted Traffic	Protocol Headers	TLS Extensions [1]	×	×	×	×	×	×	✓
		HTTPS Headers [7]	×	×	×	×	×	×	×
	Related Flows	Time Series [6]	×	×	×	×	×	×	×
		TLS Handshakes [3] Flow Statistics [16]	×	×	×	✓	×	×	✓
Plain-text and Encrypted Traffic	Network Logs	Intrusion Events [17]	✓	×	×	×	✓	×	×
		Sampled Connections [18]	✓	✓ ¹	×	✓	×	×	✓
	Traffic Features	Per-Packet Features [19]	✓	×	×	×	✓	✓	×
		Per-Flow Features [20] Flow Interaction Graph	×	×	×	✓	×	✓	×
			✓	✓	✓	✓	✓	✓	

¹ Existing multi-flow features can only represent the features of specific flows, which cannot be used to represent complicated interaction patterns among various flows.

sizes of flows, and process the interaction patterns of short and long flows separately in the graph. Specifically, it aggregates the short flows based on the similarity of massive short flows on the Internet, which reduces the density of the graph, and extracts distribution based features for long flows, which can effectively preserve flow interaction information.

We design a four-step lightweight unsupervised graph learning approach to detect encrypted malicious traffic by utilizing the rich flow interaction information maintained on the graph. First, for efficient graph learning, we analyze the connectivity of the graph by extracting the connected components and identify abnormal components by clustering the high-level statistical features. Second, we pre-cluster the edges according to the observed local adjacency in edge features, which significantly reduces the feature processing overhead and ensures realtime detection. Third, we extract critical vertices by solving a vertex cover problem to minimize the number of clustering. Finally, we cluster each critical vertex according to its connected edges, which are in the centers of the clusters produced by the pre-clustering, and thus obtain abnormal edges indicating encrypted malicious traffic.

Moreover, to quantify the benefits of the graph based flow recording of HyperVision over the existing approaches, we develop a *flow recording entropy model*, an information theory based framework that theoretically analyzes the amount of information retained by the existing data sources of malicious traffic detection systems. By using this framework, we show that the existing sampling based and event based traffic data sources (e.g., NetFlow [21] and Zeek [22]) cannot retain high-fidelity traffic information. Thus, they are unable to record flow interaction information for the detection. But the graph in HyperVision captures near-optimal traffic information for the graph learning based detection and the amount of the information maintained in the graph approaches the theoretical up-bound of the idealized data source with infinite storage according to the data processing inequality [23]. Also, the analysis results demonstrate that the graph in HyperVision achieves higher information density (i.e., amount of traffic information per unit of storage) than all existing data sources, which is the foundation of the accurate and efficient detection.

We prototype HyperVision with Intel's Data Plane Development Kit (DPDK) [24]. To extensively evaluate the prototype, we replayed 92 attack datasets including 80 new datasets collected in our virtual private cloud (VPC) with more than 1,500 instances. In the VPC, we collected 48 typical encrypted

malicious traffic, including (i) encrypted flooding traffic, e.g., flooding target links [10]; (ii) web attacks, e.g., exploiting web vulnerabilities [4]; (iii) malware campaigns, including connectivity testing, dependency update, and downloading. We observe that HyperVision achieves 13.9% accuracy improvements over five state-of-the-art methods. It detects all encrypted malicious traffic in an unsupervised manner with more than 0.86 F1, where 44 of the real-world stealthy traffic cannot be identified by all the baselines, e.g., an advanced side-channel attack exploiting the CVE-2020-36516 [14] and many newly discovered cryptojacking attacks [25]. Besides, real-world use cases show that HyperVision raises negligible false alarms, which can be easily filtered out. Moreover, HyperVision achieves on average more than 100 Gb/s detection throughput with the average detection latency of 0.29s.

In summary, the contributions of our paper are six-fold:

- We propose HyperVision, the first realtime unsupervised detection for encrypted malicious traffic with unknown patterns by utilizing the flow interaction graph.
- We develop several algorithms to build the graph that allows us to accurately capture interaction patterns among various flows.
- We design a lightweight unsupervised graph learning method to detect encrypted traffic via graph features.
- We develop a theoretical analysis framework established by information theory to show that the graph captures near-optimal traffic interaction information.
- We utilize a dimension reduction method that significantly increases detection throughput and reduces detection latency.
- We prototype HyperVision and use the extensive experiments with various real-world encrypted malicious traffic to validate its accuracy, efficiency, and robustness.

The rest of the paper is organized as follows: Section II introduces the threat model of HyperVision. Section III presents the high-level design of HyperVision. In section IV, V, and VI, we describe the detailed designs. In Section VII, we conduct the theoretical analysis. In Section VIII, we experimentally evaluate the performances. Section IX reviews related works and Section X concludes this paper.

II. THREAT MODEL AND DESIGN GOALS

We aim to develop a realtime system (i.e., HyperVision) to detect encrypted malicious traffic. It performs detection

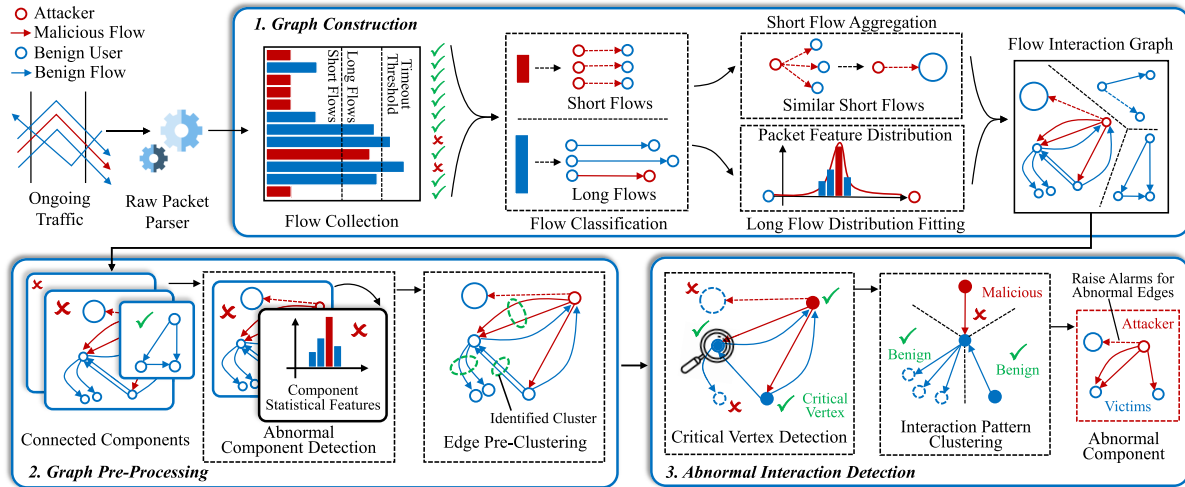


Fig. 1. The overview of HyperVision.

according to the traffic replicated by routers through port mirroring [26], which ensures that the system will not interfere with traffic forwarding. After identifying encrypted malicious traffic, it can cooperate with the existing on-path malicious traffic defenses [27], [28], [29] to throttle the detected traffic. To perform detection on encrypted traffic, we cannot parse and analyze application layer headers and payloads.

In this paper, we focus on detecting active attacks constructed with encrypted traffic. We do not consider passive attacks that do not generate traffic to victims, e.g., traffic eavesdropping [30] and passive traffic analysis [31]. According to the existing studies [12], [13], [32], [33], [34], [35], attackers utilize reconnaissance steps to probe the information of victims, e.g., the password of a victim [8], the TCP sequence number of a TLS connection [14], [36], and the randomized memory layout of a web server [37], which cannot be accessed directly by attackers due to lack of prior knowledge. Note that, these attacks are normally constructed with many addresses owned or faked by attackers.

The design goals of HyperVision are as follows: First, it should be able to achieve generic detection, i.e., detect attacks constructed with encrypted or non-encrypted traffic, which ensures that the attacks cannot evade detection by traffic encryption [3], [5]. Second, it is able to achieve realtime high-speed traffic processing, which means that it can identify whether the passing through encrypted traffic is malicious, while incurring low detection latency. Third, the performed detection by HyperVision is unsupervised, which means that it does not require any prior knowledge of encrypted malicious traffic. That is, it should be able to deal with attacks with unknown patterns, i.e., zero-day attacks, which have not been disclosed [38]. Thus, we do not use any labeled traffic datasets for ML training. These issues cannot be well addressed by the existing detection methods [2].

III. OVERVIEW OF HYPERVISION

In this section, we present the high-level design of HyperVision. Normally, patterns of each flow in the encrypted malicious traffic, i.e., single-flow patterns, may be similar to benign flows, which allow them to evade the existing detection. However, the malicious behaviors appearing in the interaction patterns between the attackers and victims will be

more distinct from the benign ones. Thus, in HyperVision, we construct a graph to represent interaction patterns among flows, and detect abnormal interactions via unsupervised graph learning. Meanwhile, it realizes generic detection by analyzing flows regardless of the traffic type and can detect encrypted and non-encrypted malicious traffic. Figure 1 shows three key parts of HyperVision, i.e., graph construction, graph pre-processing, and abnormal interaction detection.

Graph Construction. HyperVision collects network flows for graph construction. Meanwhile, it classifies the flows into short and long ones and records their interaction patterns separately for the purpose of reducing the density of the graph. In the graph, it uses different addresses as vertices that connect the edges associated with short and long flows, respectively. It aggregates the massive similar short flows to construct one edge for a group of short flows, and thus reduces the overhead for maintaining flow interaction patterns. Moreover, it records the distributions of packet features for long flows to construct the edges associated with long flows, which ensures high-fidelity recorded flow interaction patterns, while addressing the issue of coarse-grained flow features in the traditional methods [39]. We will detail how HyperVision maintains the high-fidelity flow interaction patterns in the graph in Section IV.

Graph Pre-Processing. We pre-process the built interaction graph to reduce the overhead of processing the graph by extracting connected components and cluster the components using high-level statistics. In particular, the clustering can detect the components with only benign interaction patterns accurately and thus filters these benign components to reduce the scale of the graph. Moreover, we perform pre-clustering and use the generated cluster centers to represent the edges in the identified clusters. We will detail the graph pre-processing in Section V, and further improve the efficiency of the pre-clustering via dimension reduction.

Malicious Traffic Detection Based on the Graph. We achieve unsupervised encrypted malicious traffic detection by analyzing the graph features. We identify critical vertices in the graph by solving a vertex cover problem, which ensures that the clustering based graph learning processes all edges with the minimum number of clustering. For each selected vertex, we cluster all connected edges according to their flow features and structural features that represent the flow

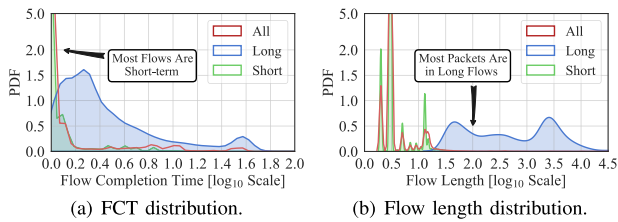


Fig. 2. The real-world flow features distribution of short and long flows.

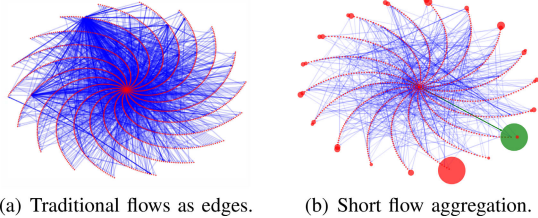


Fig. 3. HyperVision aggregates short flows to reduce the dense graph.

interaction patterns. HyperVision can identify abnormal edges in real time by computing the loss function of the clustering. We will describe the details of graph learning based detection in Section VI.

IV. GRAPH CONSTRUCTION

In this section, we present the design details of constructing the flow interaction graph that maintains interaction patterns among various flows. In particular, we classify different flows, i.e., short and long flows, and aggregate short flows. Meanwhile, we record feature distributions for long flows, respectively, for efficient graph construction. In Section VII, we will show that the graph retains the near-optimal information for detection.

A. Flow Classification

In order to efficiently analyze flows captured on the Internet, we need to avoid the dependency explosion among flows during the graph construction. We classify the collected flows into short and long flows, according to the flow size distribution [40] (see Figure 2), and then reduce the density of the graph (shown in Figure 3). Figure 2 shows the distribution of flow completion time (FCT) and flow length of the MAWI Internet traffic dataset [41]. We use first 5% (13×10^6) packets collected on a randomly selected day in Jan. 2020 to plot the figure. According to the figure, we observe that only 5.52% flows have FCT > 2.0s, which is similar to the ratios in other months, e.g., 5.34% in March and 5.18% in April. However, 93.70% packets in the dataset are long flows with only 2.36% proportion. Inspired by the observations, we apply different flow collection strategies for the short and long flows.

We poll the per-packet information from a data-plane high-speed packet parsing engine and obtain their source and destination addresses, port numbers, and per-packet features, including protocols, lengths, and arrival intervals. These features can be extracted from both encrypted and plain-text traffic for generic detection. We develop a flow classification algorithm to classify the traffic (see Algorithm 1 in Appendix A, see the Supplementary Material). It maintains a timer `TIME_NOW`, a hash table that uses `HASH(SRC, DST, SRC_PORT, DST_PORT)` as key and the collected flows indicated by the sequences of their per-packet features as values. It traverses the hash table every

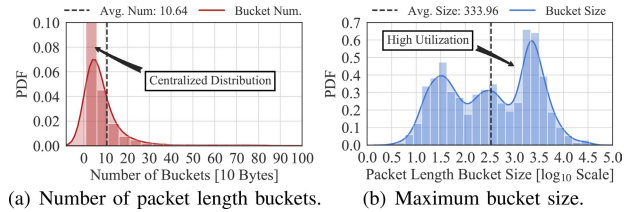


Fig. 4. The number and size of the buckets for feature distribution analysis.

JUDGE_INTERVAL second according to `TIME_NOW` and judges the flow completion when the last packet arrived before `PKT_TIMEOUT` second of `TIME_NOW`. When the flows are completed, we classify them as long flows if the flows have more than `FLOW_LINE` packets. Otherwise, we classify them as short flows. As shown in Figure 2(b), we can accurately classify short and long flows. The definitions of the hyper-parameters can be found in Table VI (see Appendix A in the Supplementary Material). Note that, we poll the state-less per-packet information from data-plane, while not maintaining flow states (e.g., a state machine [42]) on the data-plane to prevent attackers manipulating the states, e.g., side-channel attack [43] and evading detection [44].

B. Short Flow Aggregation

We need to reduce the density of the graph for analysis. As shown in Figure 3(a), the graph will be very dense for analysis if we use traditional four-tuple flows as edges, which is similar to the dependency explosion problem in provenance analysis [45], [46]. We observe that most short flows have almost the same per-packet feature sequences. For instance, the encrypted flows of repetitive SSH cracking attempts originated from specific attackers [8]. Thus, we perform the short flow aggregation to represent similar flows using one edge after the classification.

We design an algorithm to aggregate short flows (see Algorithm 2 in Appendix A, the Supplementary Material). A set of flows can be aggregated when all the following requirements are satisfied: (i) the flows have the same source and/or destination addresses, which implies similar behaviors generated from the addresses; (ii) the flows have the same protocol type; (iii) the number of the flows is large enough, i.e., when the number of the short flows reaches the threshold `AGG_LINE`, which ensures that the flows are repetitive enough. Next, we construct an edge for the short flows, which preserves a single feature sequence for all the aggregated flows, including protocols, lengths, and arrival intervals between packets. By this way, we can effectively reduce storage overhead for realtime detection. In summary, four types of edges associated with short flows exist on the graph, i.e., source address aggregated, destination address aggregated, both addresses aggregated, and without aggregation. Thus, a vertex connected to the edge can denote a group of addresses or a single address.

Figure 3 compares the graph using traditional flows as edges and our aggregated graph by using the real-world backbone traffic dataset, which is same to that used in Figure 2. The diameter of a vertex indicates the number of addresses denoted by the vertex and the depth of the color indicates the repeated edges. In Figure 3(b), we observe that the algorithm reduces 93.94% vertices and 94.04% edges. The edge highlighted in green indicates short flows (i.e., 2.38 Kpps, from PH) exploiting a vulnerability. Note that, flow aggregation is

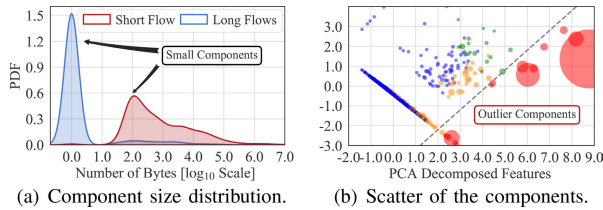


Fig. 5. The statistical features of the components.

performed upon receiving the traffic to minimize storage overhead, enabling in-memory graph maintenance for real-time detection.

C. Feature Distribution Analysis for Long Flows

Now we use histograms to represent the per-packet feature distributions (e.g., arrival intervals) of a long flow, and thus avoid preserving their long per-packet feature sequences, since the features in long flows are centrally distributed. Specifically, we maintain a hash table to construct the histogram for each per-packet feature sequence in each long flow. According to our empirical study, we set the buckets widths for packet-length and arrival interval as 10 bytes and 1 ms, respectively, to trade off between the fitting accuracy and overhead. We calculate the hash code by dividing the per-packet features by the bucket width and increase the counter indexed by the hash code. Finally, we record the hash codes and the associated counters as the histograms. Note that, the coarse-grained flow statistics, e.g., numbers of packets [39], are insufficient for encrypted malicious traffic detection [6], which also lose the flow interaction information [47]. Note that, we do not extract the distributions for short flows, since the histograms are ineffective to represent short per-packet feature sequences. Thus, we directly preserve feature sequences for short flows.

Figure 4 shows the number of the used buckets and the maximum bucket size for the long flows in the same dataset shown in Figure 2. We confirm the centralized feature distribution, i.e., most packets in the long flows have similar packet lengths and arrival intervals. Specifically, in Figure 4(a), we fit the distribution of packet length using only 11 buckets on average, and most of the buckets collect more than 200 packets (see Figure 4(b)), which demonstrate that the histogram based fitting is effective with low storage overhead. Similarly, the fitting for arrival interval uses 121 buckets on average and realizes 71 packets per bucket high utilization. Besides, we use the same method for protocol. We use the mask of protocols as the hash code and use smaller numbers of buckets to realize more efficient fitting due to the limited number of protocol types. Note that, duplicated edges between two vertices represent concurrent flows between two hosts. It implies that features of concurrent flows are individually extracted.

V. GRAPH PRE-PROCESSING

In this section, we pre-process the flow interaction graph to identify key components and pre-cluster the edges, which can enable realtime graph learning based detection against encrypted malicious traffic with unknown patterns.

A. Connectivity Analysis

To perform the connectivity analysis of the graph, we obtain the connected components by using depth-first search (DFS)

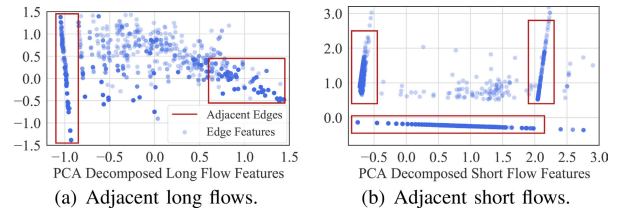


Fig. 6. The sparsity of edges in the graph feature space.

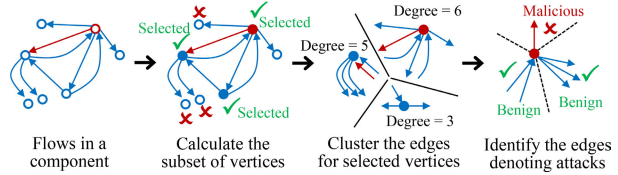


Fig. 7. Critical vertices identification via solving the vertex cover problem.

and split the graph by the components. Figure 5(a) presents the size distribution of the identified components of the MAWI traffic dataset [41] collected in Jan. 2020. We observe that most components contain few edges with similar interaction patterns. Thus, we perform a clustering on the high-level statistics for the connected components to capture the abnormal components that have over one order of magnitude clustering loss than normal components as clustering outliers. Specifically, we extract five features to profile the components, including: (i) the number of long flows; (ii) the number of short flows; (iii) the number of edges denoting short flows; (iv) the number of bytes in long flows; and (v) the number of bytes in short flows. We perform a min-max normalization and acquire the centers using the density based clustering, i.e., DBSCAN [48]. For each component, we calculate the Euclidean distance to its nearest center. We detect an abnormal component when its distance is over the 99th percentile of all the distances based on our empirical study.

Figure 5(b) shows an instance of the clustering, where the diameters indicate the scale of the traffic on the components (in the unit of bytes). We observe that most components are small, and a high ratio of huge components is classified as abnormal. All edges associated with the normal components are labeled as benign traffic, and the edges associated with the abnormal components will be further processed by the following steps.

B. Edge Pre-Clustering

Now we further need to process and pre-cluster the graph for efficient detection. As shown in Figure 5, the abnormal components in the graph have massive vertices and edges. In particular, we cannot directly apply graph representation learning, e.g., graph neural network (GNN), for realtime detection. Figure 6 shows the edges from the components in the graph structural feature space. We observe that the distribution of the edges is sparse, i.e., most edges are adjacent to massive similar edges in the feature space. To utilize the sparsity, we perform pre-clustering using DBSCAN [48] that leverages KD-Tree for efficient local search and select the cluster centers of the identified clusters to represent all edges in each cluster to reduce the overhead for graph processing.

Specifically, we extract eight and four graph structural features (see Table V in Appendix A, the Supplementary Material) for the edges associated with short and long flow, respectively, e.g., the in-degree of the source vertex of an edge associated with a long flow. These degree features of malicious traffic are significantly distinct from the benign

ones, e.g., the vertices denoting spam bots have higher out-degrees than benign clients due to their frequent interactions with servers. Then, we perform a min-max normalization for the features, and adopt a small search range ϵ and a large minimum number of points for DBSCAN clustering (see Section VIII-A for the setting of hyper-parameters) to avoid including irrelevant edges in the clusters, which may incur false positives. Moreover, some edges cannot be clustered and should be treated as outliers, which will be processed as clusters with only one edge.

VI. MALICIOUS TRAFFIC DETECTION

In this section, we detect encrypted malicious traffic by identifying abnormal interaction patterns on the graph. In particular, we cluster edges connected to the same critical vertex and detects outliers as malicious traffic (see Figure 7).

A. Identifying Critical Vertices

To efficiently learn the interaction patterns of the traffic, we do not perform clustering for all edges directly but cluster edges connected to critical vertices. For each connected component, we select a subset of all vertices in the connected component as the critical vertices according to the following conditions: (i) the source and/or destination vertices of each edge in the component are in the subset, which ensures that all the edges are connected to more than one critical vertices and clustered at least once; and (ii) the number of selected vertices in the subset is minimized, which aims to minimize the number of clustering to reduce the overhead of graph learning. Finding such a subset of vertices is an optimization problem and equivalent to the *vertex cover problem* [49], which was proved to be NP Complete (NPC). We select all edges and all vertices on each component to solve the problem. And we reformulate the problem to a Satisfiability Modulo Theories (SMT) problem that can be effectively solved by using Z3 SMT solver [50]. Since we pre-cluster the massive edges and reduce the scale of the problem (see Section V-B), the NPC problem can be solved in real time. Note that, employing thresholds to select vertices with high degrees as critical vertices is not a suitable approach. Unlike the vertex cover problem, the method does not guarantee that every edge is connected to at least one critical vertex. Consequently, any remaining unconnected edges will not be considered in clustering analysis conducted for each critical vertex.

B. Edge Feature Clustering for Detection

Now we cluster the edges connected to each critical vertex to identify abnormal interaction patterns. In this step, we use the structural features in Section V-B, and the flow features extracted from the per-packet feature sequences of short flows or the fitted feature distributions of long flows (see all the features in Appendix A, see the Supplementary Material). We employ degree based features because a vertex with a high degree denotes an abnormal user flooding massive attack flows, such as password cracking [8]. Different from existing methods, we do not analyze protocol specific features (e.g., fields in TLS headers) to achieve generic detection for various protocols. We use the lightweight K-Means algorithm to cluster the edges associated with short and long flows, respectively, and calculate the clustering loss that indicates the degree of maliciousness for malicious flow detection.

$$\text{loss}_{\text{center}}(\text{edge}) = \min_{C_i \in \{C_1, \dots, C_K\}} \|C_i - f(\text{edge})\|_2, \quad (1)$$

$$\text{loss}_{\text{cluster}}(\text{edge}) = \text{TimeRange}(\mathcal{C}(\text{edge})), \quad (2)$$

$$\text{loss}_{\text{count}}(\text{edge}) \log_2(\text{Size}(\mathcal{C}(\text{edge})) + 1), \quad (3)$$

$$\text{loss}(\text{edge}) = \alpha \text{loss}_{\text{center}}(\text{edge}) - \beta \text{loss}_{\text{cluster}}(\text{edge}) + \gamma \text{loss}_{\text{count}}(\text{edge}), \quad (4)$$

where K is the number of obtained cluster centers, C_i is the i^{th} center, $f(\text{edge})$ is the feature vector, $\mathcal{C}(\text{edge})$ contains all edges in the cluster of edge produced by pre-clustering, and TimeRange calculates the time range covered by the flows denoted by the edges.

According to Equation (4), the loss has three parts: (i) $\text{loss}_{\text{center}}$ in (1) is the Euclidean distance to the cluster centers which indicates the difference from other edges connected to the critical vertex; (ii) $\text{loss}_{\text{cluster}}$ in (2) indicates the time range covered by the cluster identified by the pre-clustering in Section V-B which implies long lasting interaction patterns tend to be benign; (iii) $\text{loss}_{\text{count}}$ in (3) is the number of flows denoted by the edges, which means a burst of massive flows implies malicious behaviors. Moreover, we used weights: α, β, γ to balance the loss terms. Finally, it detects the associated flows as malicious when the loss of an edge is larger than a threshold.

VII. THEORETICAL ANALYSIS

In this section, we develop a theoretical analysis framework, i.e., *flow recording entropy model*, to analyze the information preserved in the graph of HyperVision for graph learning based detection. The detailed analysis can be found in Appendix C, see the Supplementary Material.

A. Information Entropy Based Analysis

We develop the framework that aims to quantitatively evaluate the information retained by the exiting traffic recording modes, which decide the data representations for malicious traffic detection, by using three metrics: (i) the amount of information, i.e., the average Shannon entropy obtained by recording one packet; (ii) the scale of data, i.e., the space used to store the information; (iii) the density of information, i.e., the amount of information on a unit of storage. By using this framework, we model the graph based traffic recording mode used by HyperVision as well as three typical types of flow recording modes, i.e., (i) idealized mode that records and stores the whole per-packet feature sequence; (ii) event based mode (e.g., Zeek) that records specific events [3], [17]; and (iii) sampling based mode (e.g., NetFlow) that records coarse-grained flow information [18], [51].

We model a flow, i.e., a sequence of per-packet features, as a sequence of random variables represented by an aperiodic irreducible discrete-time Markov chain (DTMC). Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ denote the state diagram of the DTMC, where \mathcal{V} is the set of states (i.e., the values of the variables) and \mathcal{E} denotes the edges. We define $s = |\mathcal{V}|$ as the number of different states and use $\mathcal{W} = [w_{ij}]_{s \times s}$ to denote the weight matrix of \mathcal{G} . All of the weights are equal and normalized:

$$\begin{aligned} \forall 1 \leq i, j, m, n \leq s, (w_{ij} = w_{mn}) \vee (w_{ij} = 0 \vee w_{mn} = 0), \\ w_i = \sum_{j=1}^s w_{ij}, \quad 1 = \sum_{i=1}^s w_i. \end{aligned} \quad (5)$$

The state transition is performed based on the weights, i.e., the transition probability matrix $P = [P_{ij}]$, $P_{ij} = w_{ij}/w_i$. Therefore, the DTMC has a stationary distribution μ :

$$\begin{cases} \mu P = \mu, \\ 1 = \sum_{j=1}^s \mu_j \end{cases} \Rightarrow \mu_j = w_j, \quad \forall 1 \leq j \leq s. \quad (6)$$

Assume that the stationary distribution is a binomial distribution with the parameter: $0.1 \leq p \leq 0.9$ to approach Gaussian distribution with low skewness:

$$\mu \sim B(s, p) \xrightarrow{App.} \mathcal{N}(sp, sp(1-p)). \quad (7)$$

Based on the distribution, we obtain the entropy rate of the DTMC which is the expected Shannon entropy increase for each step in the state transition, i.e., the expected Shannon entropy of each random variable in the sequence, (using nat as unit, $1 nat \approx 1.44 bit$):

$$\begin{aligned} \mathcal{H}[\mathcal{G}] &= \sum_{i=1}^s \mu_i \sum_{j=1}^s p_{ij} \ln \frac{1}{p_{ij}} = - \sum_{i=1}^s \sum_{j=1}^s w_{ij} \ln w_{ij} \\ &\quad + \sum_{j=1}^s w_j \ln w_j \\ &= \ln |\mathcal{E}| - \frac{1}{2} \ln 2\pi sep(1-p). \end{aligned} \quad (8)$$

Moreover, for the real-world flow size distribution, we assume that the length of the sequence of random variables obeys a geometric distribution with high skewness, i.e., $L \sim G(q)$ with a parameter: $0.5 \leq q \leq 0.9$. \mathcal{H} , \mathcal{L} , and \mathcal{D} denote the expectation of the metrics, i.e., the amount of information, the scale of data, and the density, respectively.

Idealized Recording Mode. The idealized recording mode has infinite storage and captures optimal fidelity traffic information by recording each random variable from the sequence without any processing. Thus, the obtained information entropy of the idealized mode grows at the entropy rate of the DTMC:

$$\mathcal{H}_{Ideal} = E[L\mathcal{H}[G]] = \frac{1}{q} \ln |\mathcal{E}| - \frac{1}{2q} \ln 2\pi sep(1-p). \quad (9)$$

According to data processing inequality [23], the information retained in the idealized recording mode reaches the optimal value. It implies that processing of the observed per-packet features denoted by the random variables may incur information loss. In the following sections, we will show that the other mode incurs information loss.

We can obtain the scale of data and the density of information for the idealized recording mode as follows:

$$\mathcal{L}_{Ideal} = E[L] = \frac{1}{q}. \quad (10)$$

$$\mathcal{D}_{Ideal} = \frac{\mathcal{H}_{Ideal}}{\mathcal{L}_{Ideal}} = \mathcal{H}[G]. \quad (11)$$

Graph Based Recording Mode of HyperVision. HyperVision applies different strategies to process short and long flows for the graph construction. Let K denote the threshold for classifying the flows. When $L < K$, it collects all random variables from the sequence for short flows. Otherwise, it collects the histogram to fit the distribution for long

flows. Then, we can obtain the lower bound to estimate the information entropy in the graph of HyperVision:

$$\begin{aligned} \mathcal{H}_{H.V.} &= \frac{1 - (Kq + 1)(1 - q)^K}{q} \mathcal{H}[G] + \frac{1}{4} s(1 - q)^K \\ &\quad \times [(1 + s) \ln ps + 2 \ln 2\pi e + 2q \ln K - 2s(1 + p + \gamma)]. \end{aligned} \quad (12)$$

We can also obtain the expected data scale and the density:

$$\mathcal{L}_{H.V.} = s(1 - q)^K + \frac{1 - (Kq + 1)(1 - q)^K}{Cq}, \quad (13)$$

where C is the average number of flows denoted by an edge associated with short flows.

$$\mathcal{D}_{H.V.} = \frac{\mathcal{H}_{H.V.}}{\mathcal{L}_{H.V.}}. \quad (14)$$

Sampling Based Recording Mode. Similarly, the sampling based mode extracts and records flow statistics for the detection. We analyze the accumulative statistics (e.g. the total number of bytes) that are widely adopted [21], [39]. Let $\langle s_1, s_2, \dots, s_L \rangle$ denote the sequence of random variables, and $X_{\text{Samp.}} = \sum_{i=1}^L s_i$ indicates the flow statistic to be recorded. We can obtain a tight lower bound as an estimation for the amount of information and the other metrics as follows:

$$\mathcal{H}_{\text{Samp.}} = \mathcal{H}[X_{\text{Samp.}}] = \frac{1}{2} \ln 2\pi esp(1-p) + \frac{\ln 2}{2} q(1-q). \quad (15)$$

$$\mathcal{L}_{\text{Samp.}} = 1. \quad (16)$$

$$\mathcal{D}_{\text{Samp.}} = \mathcal{H}_{\text{Samp.}}. \quad (17)$$

Event Based Recording Mode. The event based recording mode inspects each random variable in the sequence and records events with a small probability. Since the observation that the event based methods do not generate repetitive events for a long flow with a larger s , for simplicity, we assume that the probability is $p^s \propto 1/s$. Then, we can obtain the amount of information, the scale of data, and the density of information for event based recording mode as follows:

$$\mathcal{H}_{\text{Eve.}} = -2\theta \ln \theta, \quad (18)$$

where $\theta = \frac{\zeta}{\eta}$, $\zeta = q - qp^s$, and $\eta = q - p^s(q - 1)$.

$$\mathcal{L}_{\text{Eve.}} = -\frac{p^s}{\eta}. \quad (19)$$

$$\mathcal{D}_{\text{Eve.}} = \frac{2\zeta}{p^s} \ln \theta. \quad (20)$$

B. Analysis Results

We perform numerical studies to compare the flow recording modes in real-world setting. We select three per-packet features: protocol, length, and the arrival interval (in ms) as the instances of the DTMC, then we measure the parameters of the DTMC, i.e., $|\mathcal{E}|$ and $|\mathcal{V}|$ according to the first 10^6 packets in the MAWI dataset on Jan. 2020 [41]. We also measure K , C , and estimate the geometric distribution parameter q via the second moment. We have the following three key results.

(1) *HyperVision maintains more information using the graph than the existing methods.* Figure 8 shows the results on the feasible region ($\mathcal{F} = \{0.1 \leq p \leq 0.9, 0.5 \leq q \leq 0.9\}$). We observe that HyperVision maintains at least

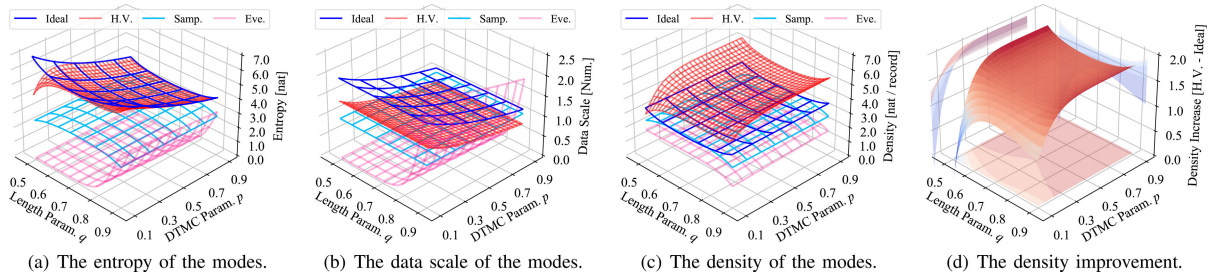


Fig. 8. The traffic information retained by different recording modes on the feasible region of the parameters.

2.37 and 1.34 times information entropy than traditional flow sampling and event based flow recording. Thus, the traditional detection methods cannot retain high-fidelity flow interaction information. Actually, they only analyze the features of a single flow, which can be evaded by encrypted traffic. According to Figure 8(b), HyperVision has 69.69% data scale of the sampling based mode. It implies that the data scale is the key challenge for the existing methods to utilize flow interaction patterns. We well address this issue by using the compact graph for maintaining the interactions among flows.

(2) *HyperVision maintains near-optimal information using the graph.* According to Figure 8(a), we observe that the information maintained by the graph almost equals to the theoretical optimum, with the difference ranging from 4.6×10^{-9} to 2.6 nat . When the parameter of the geometric distribution of L approaches 0.9, the flow information loss is larger because of the increasing ratio of long flows that incur more information loss. Figure 20 (see Appendix C in the Supplementary Material) compares the information in HyperVision and the idealized system when $q = 0.59$ and $p = 0.8$. We have similar results. The gaps between the graph mode and the optimal mode are only 0.056 and 0.021.

(c) *HyperVision has higher information density than the existing methods.* Figure 8(c) shows that HyperVision realizes 1.46, 1.54, and 2.39 times information density than the existing methods, respectively. Although the idealized system realizes the optimal amount of traffic information, the density is only 78.55% of HyperVision in the worst case, as shown in Figure 8(d). From Table VII in Appendix C, see the Supplementary Material, we find that, for all kinds of per-packet features, HyperVision can increase the density ranging between 35.51% and 47.27% due to the different recording strategies for short and long flows.

In summary, the flow interaction graph provides high-fidelity and low-redundancy traffic information with obvious flow interaction patterns, which ensures that HyperVision achieves realtime and unsupervised detection, particularly, detecting encrypted malicious traffic with unknown patterns.

VIII. EXPERIMENTAL EVALUATION

A. Experiment Setup

Implementation. We prototype HyperVision with more than 8,000 Line of Code (LOC). The prototype is compiled by gcc 9.3.0 and cmake 3.16.3. We use DPDK [24] version 19.11.9 encapsulated by libpcap++ [52] version 21.05 to implement the high-speed data-plane module. The graph construction module maintains the graph in memory for realtime detection. The graph learning module detects the encrypted malicious traffic on the interaction graph. It uses DBSCAN and K-Means in mlpack [53] (version 3.4.2) for

clustering and Z3 SMT Solver [50] (version 4.8) to identify the critical vertices.

Testbed. We deploy HyperVision on a testbed built upon DELL servers (PowerEdge R410, produced in 2012) with two Intel Xeon E5645 CPUs (2×12 cores), Ubuntu 20.04.2 (Linux 5.11.0), Docker 20.10.7, 24GB memory, one Intel 82599ES 10 Gb/s NIC, and two Intel 850nm SFP+ laser ports for optical fiber connections. We configure 6GB huge page memory for DPDK (3GB/NUMA Node) and bind 8 threads on 8 physical cores for 16 NIC RX queues to parse the per-packet features from high-speed traffic. We use 8 cores for in-memory graph construction, and 7 cores are used for graph learning, the rest one core is used as DPDK master core.

Datasets. We use real-world backbone network traffic datasets from the vantage-G of WIDE MAWI project [41] in AS2500, Tokyo Japan, Jan. ~ Jun. 2020 as background traffic. The vantage transits traffic from/to its BGP peers and providers using 10 Gb/s fiber linked to its IXP (DIX-IE), and the traffic is collected using port mirroring, which is consistent with our threat model and the physical testbed described above. We remove the attack traffic with obvious patterns in the background traffic dataset according to the rules defined by the existing studies [11], [54], [55], e.g., traffic will be detected as scanning traffic if it has scanned over 10% IPv4 addresses [11]. We generate the malicious traffic by constructing real attacks or replaying the existing traces in our testbed. Specifically, we collect malicious traffic in our virtual private cloud (VPC) with more than 1,500 instances. We manipulate the instances to perform attacks according to the real-world measurements [11], [13], [32], [54], [55], [56], [57] and the same settings in the existing studies [10], [14], [58], [59]. We classify 80 new datasets used in our experiments (see Table VIII for details) into four groups, three of which are encrypted malicious traffic:

- *Traditional brute force attack.* Although HyperVision focuses on encrypted traffic, we generate 28 kinds of traditional flooding attacks to verify its generic detection and the correctness of baselines including 18 high-rate and 10 low-rate attacks: (i) the brute scanning with the real packet rates [11]; (ii) the source spoofing DDoS with various rates [32]; (iii) the amplification attacks [55]; (iv) probing vulnerable applications [11], [60]. We collected the traffic in our VPC to avoid interference with real services.
- *Encrypted flooding traffic.* Different from the brute force flooding, the encrypted flooding is generated by repetitive attack behaviors which target specific applications: (i) the link flooding generates encrypted low-rate flows, e.g., the low-rate TCP attacks [58], [61] and the Crossfire attack [10], to congest links; (ii) injecting encrypted flows that exploits protocol vulnerabilities by flooding attack

TABLE II
THE AVERAGE ACCURACY ON THE GROUPS OF DATASETS

Method	Metric	Traditional Attacks	Flooding Enc. Traffic	Enc. Web Attacks	Malware Traffic	Overall
Jaquen	AUC	0.913 \uparrow 7%	0.782 \uparrow 19%	N/A ¹	N/A	0.867 \uparrow 12%
	F1	0.819 \uparrow 16%	0.495 \uparrow 46%	N/A	N/A	0.705 \uparrow 26%
FlowLens	AUC	0.939 \uparrow 4%	0.757 \uparrow 22%	0.685 \uparrow 30%	0.768 \uparrow 22%	0.752 \uparrow 36%
	F1	0.799 \uparrow 18%	0.651 \uparrow 29%	0.384 \uparrow 59%	0.411 \uparrow 57%	0.451 \uparrow 41%
Whisper	AUC	0.951 \uparrow 3%	0.932 \uparrow 4%	0.958 \uparrow 2%	0.648 \uparrow 34%	0.752 \uparrow 23%
	F1	0.705 \uparrow 27%	0.461 \uparrow 50%	0.546 \uparrow 42%	0.357 \uparrow 62%	0.407 \uparrow 57%
Kitsune	AUC	0.748 \uparrow 24%	- ²	0.759 \uparrow 22%	-	0.751 \uparrow 23%
	F1	0.419 \uparrow 57%	-	0.366 \uparrow 61%	-	0.402 \uparrow 58%
DeepLog	AUC	0.716 \uparrow 27%	0.621 \uparrow 26%	0.767 \uparrow 22%	0.653 \uparrow 34%	0.666 \uparrow 32%
	F1	0.513 \uparrow 47%	0.508 \uparrow 45%	0.572 \uparrow 40%	0.628 \uparrow 34%	0.597 \uparrow 37%
H.V.	AUC	0.988 \uparrow 8%	0.974 \uparrow 4%	0.985 \uparrow 2%	0.993 \uparrow 29%	0.988 \uparrow 13%
	F1	0.978 \uparrow 19%	0.927 \uparrow 42%	0.957 \uparrow 67%	0.970 \uparrow 54%	0.960 \uparrow 36%

¹ The results are N/A because Jaquen is designed for detection of volumetric attacks.

² - means that the average AUC is lower than 0.60, which is nearly the result of random guessing.

traffic and inject packets into the channel [14], [59], [62]; (iii) the password cracking performs slow attempts to hijack the encrypted communication protocols [8], [63]. We perform SSH cracking in the VPC with the scale of SSH servers in the ASes reachable to AS2500.

- **Encrypted web malicious traffic.** Web malicious traffic is normally encrypted by HTTPS. We collect the traffic generated by seven widely used web attacks including automatic vulnerabilities discovery (including XSS, CSRF, various injections) [4], SSL vulnerabilities detection [64], and crawlers. We also collect the SMTP-over-TLS spam traffic that lures victims to visit the phishing sites [15].
- **Malware generated encrypted traffic.** The traffic of malware campaigns is low-rate and encrypted, e.g., malware component update or delivery [65], command and control (C&C) channel [18], and data exfiltration [5]. We use the malware infection statistics published in 2020 [57] and probed active addresses from the adopted vantage [66], [67] to estimate the number of visible victims. We use the same number of instances to replay public malware traffic datasets [68], [69] to mimic malware campaigns, which is similar to the existing study [70].

The malicious traffic is replayed with the background traffic datasets on the physical testbed simultaneously according to their original packet rates [41] which is the same as the existing studies [38], [51], [71]. Specifically, each dataset contains 12~15 million packets and the replay lasts 45s and the first 75% time does not contain malicious traffic for collecting flow interactions and training the baselines. Note that, the rates of the encrypted attack flows in our datasets are only 0.01 ~ 8.79 Kpps which consume only 0.01% ~ 0.72% bandwidth. We will show that these stealthy attacks evade most baselines.

To eliminate the impact of the dataset bias, we also use 12 existing datasets including the Kitsune datasets [19], the CIC-DDoS2019 datasets [72], and the CIC-IDS2017 datasets [73], which are collected in the real-world. In particular, the traffic in two CIC datasets [72], [73] lasts 6~8 hours under multiple attacks, which aims to verify the long-run performances of HyperVision. Besides, we generate 48 evasion attack datasets for robustness analysis according to a recent study [38]. These details can be found in Section VIII-B.

Baselines. We use five state-of-the-art generic malicious traffic detection methods as baselines:

- **Jaquen** (*sampling based recording and signature based detection*). Jaquen [51] uses Sketches to obtain flow statistics and applies the threshold based detection. We prototype Jaquen on the testbed, and adjust the signatures for each statistic and each attack to obtain the best accuracy.
 - **FlowLens** (*sampling based recording and ML based detection*). FlowLens [20] uses sampled flow distribution and supervised learning, i.e., random forest. We use the hyper-parameter setting with the best accuracy used in the paper to retrain the ML model.
 - **Whisper** (*flow-level features and ML based detection*). Whisper [38], [74] extracts the frequency domain features of flows and uses clustering to learn the features. We deploy Whisper on the physical testbed without modifications and then retrain the clustering model.
 - **Kitsune** (*packet-level features and DL based detection*). Kitsune extracts per-packet features and uses autoencoders to learn the features which is an unsupervised method [19]. We use its default hyper-parameters and retrain the model.
 - **DeepLog** (*event based recording and DL based detection*). DeepLog is a general log analyzer using LSTN RNN [17]. We use the logs of connections for detection and its original hyper-parameter setting to achieve the best accuracy. Note that, in the baselines above, we do not include DPI-based encrypted malicious traffic detection because they are unable to investigate encrypted payloads [9]. Also, we do not compare the task-specific detection methods [6], [7] because they cannot achieve acceptable detection accuracy. Features in FlowLens, Kitsune, and Whisper are similar to them, e.g., flow features [7], packet header features [3], and time-series [6].
- Metrics.** We mainly use AUC (i.e., the area under a receiver operating characteristic curve) and F1 score because they are most widely used in the literature [17], [18], [19], [37], [38], [75], [76]. Also, we use other six metrics to validate the improvements of HyperVision, which can be found in Section VIII-B.

Hyper-parameter Selection. We conduct four-fold cross validation to avoid overfitting and hyper-parameter bias. Specifically, the datasets are equally partitioned into four subsets. Each subset is used once as a validation set to tune the hyper-parameters via the empirical study and the remaining three subsets are used as testing sets. Finally, four results are averaged to produce final results. Moreover, our ablation study shows that the different threshold settings incur at most 5.2% accuracy loss. Therefore, the hyper-parameter selection has limited impacts on the detection results.

B. Accuracy Evaluation

Table II summarizes the detection accuracy and the improvements of HyperVision over the existing methods. In general, HyperVision achieves average F1 ranging between 0.927 and 0.978 and average AUC ranging between 0.974 and 0.993 on the 80 datasets, which are 35% and 13% improvements over the best accuracy of the baselines. In 44 datasets, none of the baselines achieves F1 higher than 0.80, which means that they are not effective to detect the attacks. Due to the page limits, we do not show the failed detection results of these baselines.

Traditional Brute Force Attacks. First, we measure the performance of the baselines by using the flooding attacks

TABLE III
DETECTION ACCURACY OF HYPERVISION AND THE BASELINES ON TRADITIONAL BRUTE FORCE ATTACKS

Method	Metric	Brute Scanning							Amplification Attack						Source Spoofing DDoS				
		ICMP	NTP	SSH	SQL	DNS	HTTP	HTTPS	NTP	DNS	CharG.	SSDP	RIPv1	Mem.	CLDAP	SYN	RST	UDP	ICMP
Jaquen	AUC	0.9478	0.9989	0.9706	0.9851	0.9989	0.9774	0.9988	0.9822	0.9590	0.9860	0.9907	0.9011	0.9586	0.9537	0.9976	0.9985	0.9682	0.9995
	F1	0.9710	0.9356	0.9835	0.9924	0.9965	0.9884	0.9299	0.9457	0.8816	0.7986	0.7054	0.6549	0.8500	0.7931	0.9614	0.9236	0.5603	0.9861
FlowLens	AUC	0.9906	0.9021	0.9961	0.9993	0.9985	0.9874	0.9226	0.9784	0.8001	0.9998	0.9907	0.9833	0.9786	0.9993	0.9912	0.9918	0.9999	0.6351
	F1	0.9181	0.6528	0.8899	0.9996	0.9992	0.9936	0.9572	0.9794	0.7127	0.9991	0.8918	0.9889	0.9691	0.9986	0.8638	0.8173	0.9990	0.2632
Whisper	AUC	0.9499	0.9796	0.9562	0.9811	0.9832	0.9658	0.9827	0.9125	0.9645	0.8489	0.9662	0.9761	0.8954	0.9402	0.9563	0.9658	0.8956	0.9489
	F1	0.7004	0.7585	0.8869	0.7022	0.6748	0.7182	0.7489	0.8248	0.8435	0.4686	0.6195	0.6396	0.6956	0.8620	0.7587	0.8778	0.4857	0.4192
Kitsune	AUC	0.4522	0.7252	- ²	0.7439	0.7228	0.7380	0.9614	0.7340	0.9994	0.9998	0.9989	0.4343	0.3993	0.7592	0.6210	0.4086	0.8534	0.7913
	F1	- ¹	0.3459	-	0.5033	0.4923	0.4798	0.4878	0.4461	0.5031	0.4609	0.4360	-	-	0.3838	0.3361	-	0.4539	0.4153
DeepLog	AUC	0.6717	0.8232	0.8377	0.6518	0.8261	0.6617	0.5545	0.7475	0.7428	0.7462	0.7458	0.7487	0.7480	0.7483	0.7564	0.2470	0.7012	0.7521
	F1	0.3566	0.4178	0.5266	0.2695	0.4050	0.2668	0.3653	0.5108	0.7201	0.5705	0.4313	0.3368	0.3321	0.3424	0.6074	-	0.4370	0.3428
H.V.	AUC	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9999	0.9998	0.9989	0.9998	0.9969	0.9999	0.9999	0.9999	0.9996	0.9928
	F1	0.9939	0.9928	0.9960	0.9932	0.9831	0.9808	0.9892	0.9998	0.9998	0.9992	0.9956	0.9984	0.9983	0.9996	0.9993	0.9571	0.9981	0.9295

¹ We highlight the best accuracy in • and the worst accuracy in •. We mark - for the F1 when the AUC is lower than 0.50, which is the accuracy of random guessing.
² Kitsune did not finish the detection within 90 min (i.e., meaningless for defenses). And H.V. is short for HyperVision.

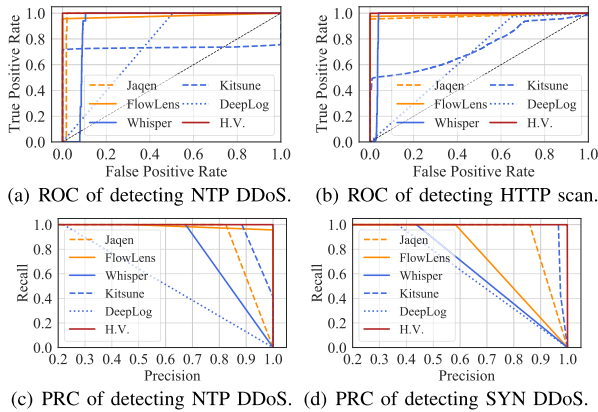


Fig. 9. ROC and PRC of HyperVision and all the baselines.

with short flows. Although HyperVision is designed for encrypted malicious traffic detection, we find that it can also detect traditional attacks accurately. The results are shown in Table III. HyperVision has 0.992 ~ 0.999 AUC and 0.929 ~ 0.999 F1, which achieves at most 13.4% and 1.3% improvement of F1 and AUC over the best performance of the baselines. The ROC and PRC results are illustrated in Figure 9. According to Figure 9(a) and 9(b), we observe that HyperVision has less false positives while achieving similar accuracy. Figure 9(c) and Figure 9(d) show that the PRC of HyperVision is largely better than the baselines, which means that it has a higher precision when all methods reach the same recall.

Second, by comparing HyperVision with Jaquen, we can see that HyperVision can realize higher accuracy (i.e., a 19.4% F1 improvement) than Jaquen with the best threshold set manually. That is, the unsupervised method allows reducing manual design efforts. Moreover, it has 56.3% AUC improvement over the typical supervised ML based method (FlowLens). Note that, we assume that HyperVision cannot acquire labeled datasets for training, which is more realistic. Also, it outperforms Whisper with 11.6% AUC, which is an unsupervised detection in high-speed network. We observe that Kitsune and DeepLog have lower accuracy because they cannot afford high-speed backbone traffic.

Third, we measure the detection accuracy of probing vulnerable applications. As shown in Figure 10, we see that HyperVision can detect the low-rate attacks with 0.920 ~ 0.994 F1 and 0.916 ~ 0.999 AUC under 6 ~ 268 attackers with 17.6 ~ 97.9 Kpps total bandwidth. It also achieves at most 46.8% F1 and 27.3% AUC improvements over the baselines that have a more significant accuracy decrease than the high-rate attacks. For example, FlowLens only achieves averagely 0.684 F1, which is only 77% under the high-rate attacks. Although Jaquen can be deployed on programmable switches, its thresholds are invalidated by the low-rate attacks. And Whisper is unable to detect the attacks with two datasets. Moreover, Kitsune and DeepLog cannot detect the attacks because of the low rate of malicious packets ($\leq 1.2\%$).

The reason why HyperVision can detect the slow probing while maintaining the similar accuracy to the high-rate attacks is that the graph preserves flow interaction patterns. Although the flows from a single attacker are slow, e.g., at least 244 pps, HyperVision can record and analyze their interaction patterns. Specifically, each flow in the stealthy attack traffic can be represented by an edge in the graph, while the vertices in the graph indicate the addresses generating the traffic. Thus, the traffic can be captured by identifying vertices with large out-degrees (i.e., a large number of edges). Moreover, the brute force attacks validate that our method is effective to capture the DDoS traffic because it utilizes the short flow aggregation to construct the edge associated with short flows and avoids inspecting each short spoofing flow. Besides, the experiment results also show that the critical vertices denote the addresses of major active flows, e.g., web servers, DNS servers, and scanners. Note that, we exclude the results of the baselines that cannot detect encrypted traffic with lower rates in the following sections due to the page limits.

Encrypted Flooding Traffic. Figure 11 shows the detection accuracy under flooding attacks using encrypted traffic. Generally, HyperVision achieves 0.856 ~ 0.981 F1 and 0.917 ~ 0.998 AUC, which are 58.7% and 25.3% accuracy improvements over the baselines that can detect such attacks. Specifically, as shown in Figure 11(a) and 11(b), we observe that HyperVision can accurately detect the link flooding traffic consists of various encrypted traffic with different parameters. For instance, it can detect the Crossfire attack using HTTPS web requests generated by different sizes of botnets [10]

	SMTp	NetBios	Telnet	VLC	SNMP	RDP	HTTP	DNS	ICMP	SSH
Jaen	0.9864	0.8117	0.6214	0.8829	0.9864	0.6280	-	0.9975	0.9674	0.7123
FlowLens	0.9649	0.9615	0.9693	0.9518	0.9649	0.9639	-	0.9480	0.9688	0.9226
Whisper	0.9611	0.9553	0.9672	0.9540	0.9611	0.9594	-	0.9530	-	0.9549
Kitsune	0.9484	0.9363	0.9410	0.9309	0.9484	-	0.8526	0.8539	0.8959	-
DeepLog	0.6501	0.6504	0.6496	0.8515	0.6501	0.6496	0.6751	0.8486	0.6503	0.8248
H.V.	0.9707	0.9587	0.9439	0.9902	0.9999	0.9751	0.9161	0.9706	0.9882	0.9868

(a) AUC of detecting probing vulnerable application.

Jaen	0.8354	0.5484	0.4740	0.5333	0.8354	0.4565	-	0.9748	0.9616	0.4997
FlowLens	0.6211	0.7040	0.8569	0.6416	0.6211	0.8439	-	0.6561	0.8861	0.9572
Whisper	0.7048	0.8013	0.8583	0.7633	0.7048	0.8528	-	0.8030	-	0.7525
Kitsune	0.3232	0.3724	0.4601	0.3710	0.3232	-	0.5236	0.2406	0.4909	-
DeepLog	0.3569	0.6211	0.7046	0.8333	0.3569	0.7068	0.7128	0.8530	0.7176	0.6645
H.V.	0.9207	0.9469	0.9664	0.9790	0.9944	0.9791	0.9471	0.9332	0.9869	0.9323

(b) F1 of detecting probing vulnerable application.

Fig. 10. Heatmap of accuracy for probing vulnerabilities.

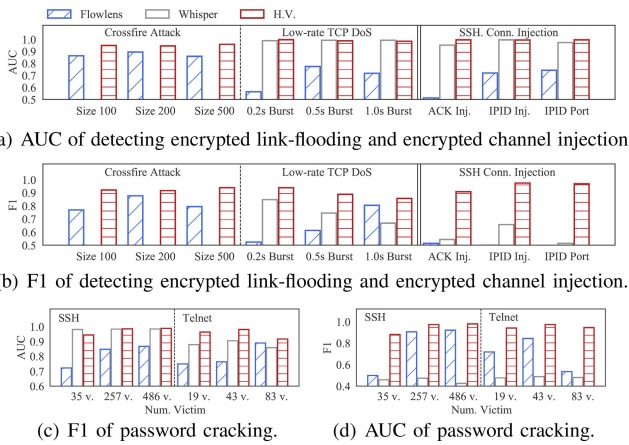


Fig. 11. Detection accuracy of encrypted flooding traffic.

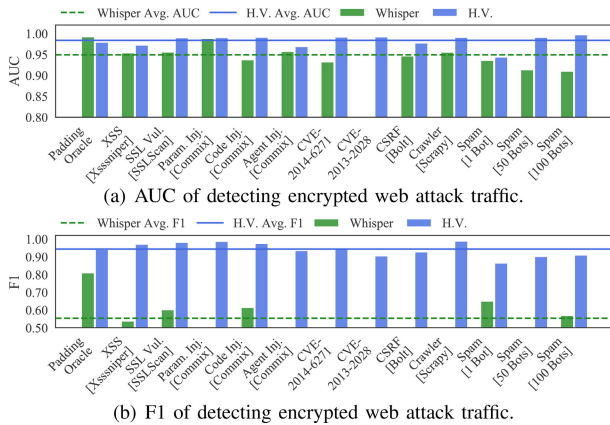


Fig. 12. Accuracy of encrypted web attack traffic detection.

with at most 0.939 F1. The massive web traffic generated by bots, which is low-rate (≤ 4 Kbps) and encrypted, evades the detection of Whisper and FlowLens ($F1 \leq 0.8$). As shown in Figure 13(a), HyperVision can detect the attack efficiently by splitting the botnet clusters into a single connected component to exclude the interference from the similar benign web traffic, where the inner layer denotes botnets and the outer denotes decoy servers.

Moreover, we find that HyperVision can detect low-rate TCP DoS attacks that use burst encrypted video traffic for at most 0.995 AUC and 0.938 F1. Although Whisper has slightly better AUC in some cases, we find that it cannot

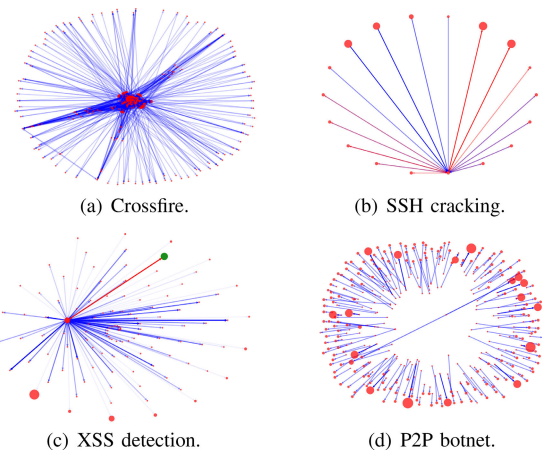


Fig. 13. Subgraph with various encrypted malicious traffic.

TABLE IV
DETECTION ACCURACY UNDER VARIOUS EVASION ATTACKS

Evasion Attacks		Obfuscation		Reduced Rates		Faked Lengths	
Dataset		F1	$\Delta F1$	F1	$\Delta F1$	F1	$\Delta F1$
Vulnerability Scanning	NTP	0.9995	$\blacktriangledown 0.04\%$	0.9995	$\blacktriangledown 0.04\%$	0.9993	$\blacktriangledown 0.06\%$
	SSH	0.9983	$\blacktriangledown 0.16\%$	0.9983	$\blacktriangledown 0.16\%$	0.9943	$\blacktriangledown 0.56\%$
	SQL	0.9983	$\blacktriangledown 0.16\%$	0.9983	$\blacktriangledown 0.16\%$	0.9943	$\blacktriangledown 0.56\%$
	DNS	0.9994	$\blacktriangledown 0.05\%$	0.9993	$\blacktriangledown 0.06\%$	0.9984	$\blacktriangledown 0.15\%$
	HTTP	0.9906	$\blacktriangledown 0.93\%$	0.9986	$\blacktriangledown 0.13\%$	0.9946	$\blacktriangledown 0.53\%$
	HTTPS	0.9987	$\blacktriangledown 0.12\%$	0.9987	$\blacktriangledown 0.12\%$	0.9946	$\blacktriangledown 0.53\%$
Amplification Attack	NTP	0.9996	$\blacktriangledown 0.03\%$	0.9999	$\blacktriangle 0.00\%$	0.9999	$\blacktriangle 0.00\%$
	DNS	0.9999	$\blacktriangle 0.00\%$	0.9999	$\blacktriangle 0.00\%$	0.9999	$\blacktriangle 0.00\%$
	CharG.	0.9387	$\blacktriangledown 6.12\%$	0.9999	$\blacktriangle 0.00\%$	0.9995	$\blacktriangledown 0.04\%$
	SSDP	0.9999	$\blacktriangle 0.00\%$	0.9999	$\blacktriangle 0.00\%$	0.9999	$\blacktriangle 0.00\%$
	RIPv1	0.9999	$\blacktriangle 0.00\%$	0.9999	$\blacktriangle 0.00\%$	0.9999	$\blacktriangle 0.00\%$
	Mem.	0.9999	$\blacktriangle 0.00\%$	0.9999	$\blacktriangle 0.00\%$	0.9550	$\blacktriangledown 4.49\%$
Spoofing DDoS	CLDAP	0.9999	$\blacktriangle 0.00\%$	0.9999	$\blacktriangle 0.00\%$	0.9999	$\blacktriangle 0.00\%$
	SYN	0.9999	$\blacktriangle 0.00\%$	0.9999	$\blacktriangle 0.00\%$	0.9999	$\blacktriangle 0.00\%$
	RST	0.9999	$\blacktriangle 0.00\%$	0.9999	$\blacktriangle 0.00\%$	0.9999	$\blacktriangle 0.00\%$
UDF	0.9999	$\blacktriangle 0.00\%$	0.9999	$\blacktriangle 0.00\%$	0.9999	$\blacktriangle 0.00\%$	
Average		0.9995	$\blacktriangledown 0.01\%$	0.968	$\blacktriangledown 0.00\%$	0.9956	$\blacktriangledown 0.01\%$

achieve high accuracy on all scenarios. As a result, it has only 55.5% AUC in the worse case. Moreover, HyperVision can aggregate the short flows in the SSH connection injection attacks and achieves more than 0.95 F1. The attacks exploiting protocol vulnerabilities realize low-rate packet injection and evade the detection of FlowLens (i.e., $AUC \leq 0.774$, $F1 \leq 0.513$). Figure 11(c) and 11(d) illustrate that HyperVision can identify slow and persisted password attempts for the channels with over 0.881 F1 and 0.917 AUC, which are 1.19 and 1.28 times improvements over FlowLens and Whisper. The reason is that HyperVision maintains the interaction patterns of attackers using the graph, e.g., the massive short flows for login attempts shown as red edges in Figure 13(b).

Encrypted Web Malicious Traffic. Figure 12 presents the detection accuracy of the encrypted traffic generated by various web vulnerabilities discovery. HyperVision achieves 0.985 average AUC and 0.957 average F1 (i.e., 2.8% and 75.2% increase compared to Whisper). The flow based ML detection cannot detect web encrypted malicious traffic because the traffic has single-flow patterns that are almost same to benign web access flows. HyperVision can accurately detect the encrypted web malicious traffic, because, as shown in Figure 13(c), it captures the traffic from the frequent

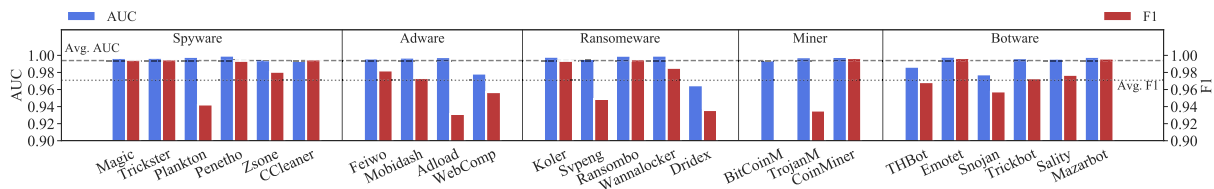


Fig. 14. HyperVision can detect various encrypted malware traffic.

interactions as the edges associated with long flows, and identifies the malicious traffic (denoted by red edges) generated by the attacker (denoted by the green vertex) by clustering the edges associated with benign web traffic that are connected to the same critical vertex (denoted by the red solid vertex).

Encrypted Malware Traffic. We show the detection accuracy of encrypted malware traffic in Figure 14. Note that, the encrypted malware traffic is hard to detect for the baselines because it is slow and persistent. However, HyperVision accurately detects the malware campaigns with at least 0.964 AUC and 0.891 F1. Specifically, it captures the C&C servers of spyware for exfiltration as abnormal critical vertices that are connected by massive infected hosts in the graph. As a result, it detects the encrypted malicious traffic of the malware with at least 0.942 F1. For example, to detect Sality P2P botnet shown in Figure 13(d), HyperVision collects the interactions among similar P2P bots, aggregates the encrypted short flows as edges, and finally clusters the edges with higher loss than benign interaction patterns. Similarly, it can capture the static servers of adware, malware component delivery servers, the infected miner pools as abnormal vertices. Note that, the low-rate malicious flows (at least 0.814 pps) are represented as the edges associated with short flows connected to critical vertices. Meanwhile, the massive long flows with almost 100% encrypted packet proportion are represented as the edges associated with long flows to the vertices. Therefore, a critical vertex connected with the edges indicates the malware campaign that is significantly different from benign vertices with large degrees, e.g., benign websites.

Robustness Analysis According to a recent study [38], we validate the robustness of HyperVision against three typical evasion strategies: (i) Obfuscation: attackers inject benign TLS traffic and UDP video traffic into attack traffic. We set the ratio of malicious packets and benign packets to 1:4, which can evade most existing detection methods [19], [20]. (ii) Reducing sending rates: attackers reduce their sending rates by 50%. (iii) Manipulating traffic features: attackers fake packet length to mimic the benign flows used in (i). According to the three strategies, we generate 48 evasion attack datasets based on 16 traditional attacks.

Table IV illustrates that HyperVision incurs negligible accuracy decrease when detecting the evasion attacks. In particular, the decreased accuracy is bounded by 6.12% F1. And the average accuracy decreases incurred by the three evasion strategies are 0.005%, 0.001%, and 0.005%, respectively. Therefore, HyperVision is robust against all the evasion strategies. The reason why these evasion techniques decrease negligible accuracy is that they only manipulate the patterns of a single flow. HyperVision can still detect the evasion attacks by learning the interaction patterns among various flows.

Detection Accuracy of Other Datasets. We use 12 existing datasets to eliminate the impact of dataset bias. From Table IX,

we can see that HyperVision achieves 7.8%, 11.0%, 5.1% F1 improvements over the best accuracy of the baselines on Kitsune datasets [19], CIC-IDS2017 datasets [73], and CIC-DDoS2019 datasets [72], respectively. In addition, we validate the long-run performance of HyperVision, by using the CIC datasets [72], [73]. Specifically, the experiments show that HyperVision achieves over 0.95 F1 and 0.99 AUC in long-run detection (6~8 hours). The results also verify that the accumulation of detection errors cannot interfere with HyperVision, and HyperVision can detect multiple attacks simultaneously even in the presence of attacks with changed addresses [73]. Moreover, the memory consumption of the compact graph is bounded by 15.6 GB.

Other Metrics for Detection Accuracy. We use other six metrics to evaluate the performance of HyperVision. Figure 15 depicts the results. In general, HyperVision has at least 34.3%, 11.2%, 40.9%, and 9.4% improvements on F2, accuracy, precision, and recall, respectively. And we conclude that HyperVision can achieve both high precision and recall (≥ 0.96). From the metrics of detection errors, we observe that HyperVision reduces 4.1 times and 160.1 times EER and FPR over the existing methods.

Impacts of False Positives. Our experiments show that the number of false positives (FPs) can be bounded by 41.36 FPs per minute, which is 10.07, 10.33, and 55.46 times lower than FlowLens, Jaqen, and Kitsune, respectively. For instance, when detecting encrypted flooding traffic, the average number of FPs raised by HyperVision is only 4.39 alarms per minute, which is 17.39 times lower than the lowest value of the baselines. The reason why HyperVision rarely rises FPs is that, it excludes majorities of benign flows in the graph pre-processing module (see Section V-A), which prevents the benign traffic from triggering FPs.

In addition, we find that these FPs can be easily filtered out by analyzing graph features. Specifically, we apply thresholds on four graph features, i.e., the in- and out-degree of the source/destination vertices of an edge. In practice, we manually set the thresholds according to the FPs triggered by 25% traffic in the datasets. The results show that the thresholds can filter out 97.00%~99.99% FPs so that the number of FPs can be bounded by 1.3 FPs per minute. Thus, HyperVision does not suffer from the FP issue even if it utilizes unsupervised learning.

C. Performance Results

Throughput. We truncate the packets to the first 200 bytes on the physical testbed and increase the sending rates until the graph construction module reaches maximum throughput. Figure 16 shows the throughput of the graph construction and the detection. Figure 16(a) presents the distribution of average throughput within a 1.0s time window. We observe that HyperVision constructs the graph for 28.21 Gb traffic per second. Figure 16(b) presents the maximum throughput in each time window with all the backbone traffic datasets

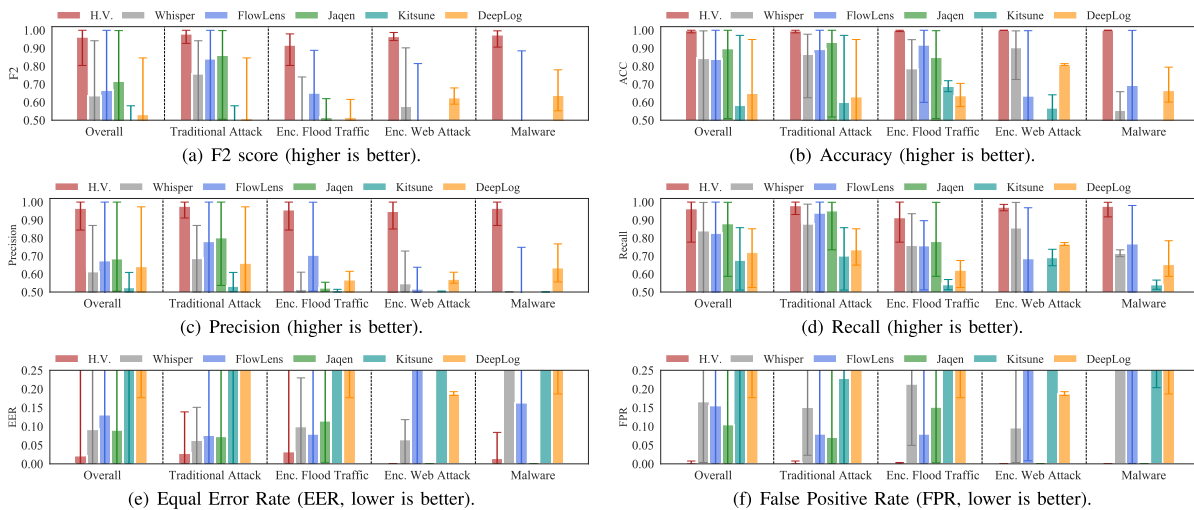


Fig. 15. Other six metrics for detection accuracy.

used in the experiments. HyperVision achieves 32.43 ~ 39.71 peak throughput on average. Moreover, we measure the throughput of the graph learning module, which inspects flow interactions. According to Figure 16(c), we observe that it can analyze 121.14 Gb traffic per second on average. Note that, the detection throughput is 4.2 times higher than the construction so that the detection can analyze the recorded traffic iteratively to consider the past interaction information. We observe that the average throughput exhibits a bimodal distribution. The peak of low throughput (around 75 Gb/s) is caused by lacking the information on the graph for analyzing during cold start stages. Figure 16(d) illustrates the throughput when the performance of the system is stable. We observe that it achieves 80.6 ~ 148.9 Gb/s throughput. Note that, the throughput on Apr. and Jun. datasets is lower because of their low traffic volume.

Latency. We measure the latency caused by graph construction and detection. Figure 17(a) presents the PDF of the maximum latency for constructing each edge within a 1.0s window. We observe that HyperVision has 1.09s ~ 1.04s average construction latency with an upper bound of 1.93s. The distribution is a significant bimodal one because the receive side scaling (RSS) on the Intel NIC is unbalanced on the threads. The light-load threads have only 0.75s latency. We analyze the composition of the latency in Figure 17(b) (where the error bar is 10th and 90th percentile) and find that the flow classification, short flow analysis, and long flow analysis share 50.95%, 35.03%, and 14.0% latency, respectively. We measure the average detection latency. Figure 17(c) shows that the learning module has a 0.83s latency on average with a 99th percentile of 4.48s. We also analyze the latency in each step (see Figure 17(d)). We see that 75.8% of the latency comes from pre-clustering (i.e., 0.66s on average). However, the pre-clustering step reduces the processing overhead of the subsequent processing, i.e., selecting critical vertex and clustering, for 5.5×10^{-3} s (0.64%) and 3.4×10^{-3} s (0.40%).

Resource Consumption. Figure 18(a) presents the memory usage of HyperVision. Note that, the DPDK huge pages require 6GB memory and thus we measure the consumption when the usage reaches 6GB. We observe that the increasing rate of memory for maintaining the graph is only 13.1 MB/s. Finally, HyperVision utilizes 1.78 GB memory to maintain the flow interaction patterns extracted from 2.82 TB ongoing

traffic. HyperVision incurs low memory consumption because the feature extraction for long flow and short flow aggregation make the in-memory graph compact which ensures low-latency detection and long-term recording. Moreover, the memory consumption of the learning algorithm is 1.452 ~ 1.619 GB. HyperVision can export the graph to disk for forensic analysis. Figure 18(b) shows the storage used for recording the first 45s traffic of the MAWI dataset by different methods, i.e., HyperVision, event based network monitors (i.e., Suricata [77] and Zeek [22]), and raw packet headers. We observe that HyperVision achieves 8.99%, 55.7%, 98.1% storage reduction over the baselines, respectively. Meanwhile, our analysis shows that HyperVision retains more traffic information than the existing tools (see Section VII). Thus, the graph based analysis is more efficient than existing tools.

Benefits of Dimension Reduction. From Figure 17(d), we observe that the DBSCAN clustering based graph pre-processing incurs over 96.8% latency which implies the clustering is the bottleneck of the whole system. Therefore, we leverage a dimension reduction method to reduce the overhead of DBSCAN. In particular, we reduce the dimension of the graph feature from 12 to 2 by using PCA before conducting the DBSCAN clustering. By this mean, we significantly reduce the scale of data and thus improve the detection efficiency [78].

Figure 19 shows the improvements by the dimension reduction. Particularly, the dimension reduction method increases the average throughput by 27.34% (i.e., from 121.4 Gb/s to 154.6 Gb/s) on the Jan. 2020 dataset. Also, from Figure 19(a), we observe that the average detection latency is 0.292s when enabling PCA, i.e., the dimension reduction can reduce 64.39% detection latency. Besides, we validate that applying dimension reduction incurs negligible accuracy decrease, which can be bounded by 2.3% F1 decrease.

IX. RELATED WORK

Graph Based Anomaly Detection. Graph based structures have been used for task-specific traffic detection. These methods heavily rely on DPI and thus cannot be applied to detect encrypted traffic [6]. Kwon et al. analyzed the download relationship graph to identify malware downloading [79], which is similar to WebWitness [80]. Eshete et al. constructed HTTP interaction graphs to detect malware static

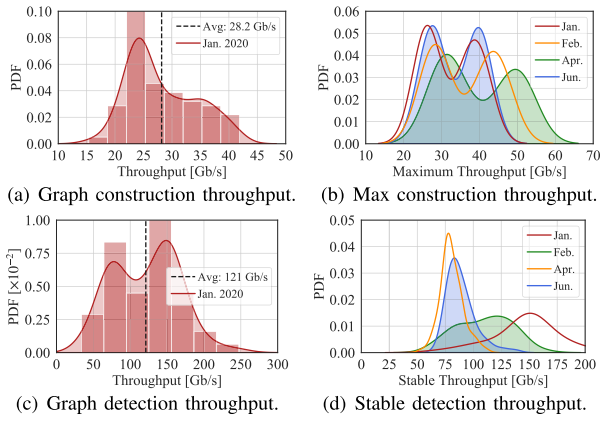


Fig. 16. Throughput of graph construction and detection.

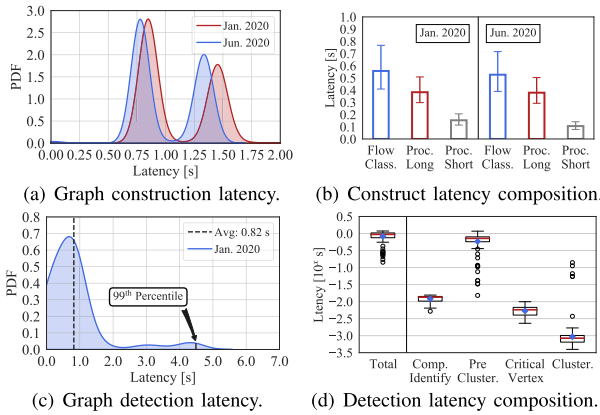


Fig. 17. Latency of graph construction and detection.

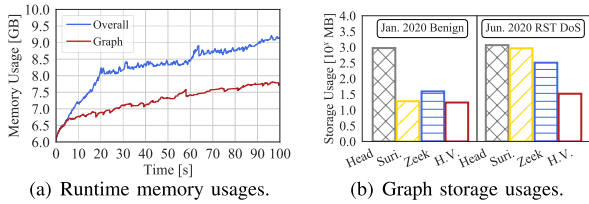


Fig. 18. Hardware resource usages of HyperVision.

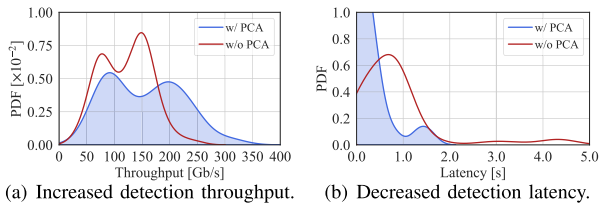


Fig. 19. Improved throughput and latency by dimension reduction.

resources [13], and Invernizzi et al. used a graph constructed from plain-text traffic to identify malware infrastructures [81]. Different from these works, HyperVision constructs the interaction graph without parsing specific application layer headers and thus achieves task-agnostic encrypted traffic detection. Note that, the provenance graph based attack forensic analysis [45], [46] is orthogonal to our traffic detection.

ML Based Malicious Traffic Detection. ML based detection can detect zero-day attacks [82] and achieve higher accuracy than the traditional signature based methods [42], [83], [84], [85]. For example, Barradas et al. developed Flowlens to extract flow distribution features on data-plane and

detect attacks by applying random forest [20]. Stealthwatch detected attacks by analyzing flow features extracted from NetFlow [1]. Mirsky et al. developed Kitsune to learn the per-packet features by adopting auto-encoders [19]. Zhou et al. installed decision tree models on Intel Tofino programmable switches, enabling line-rate traffic detection [86]. Siracusano et al. deployed binary neural networks on SmartNICs, and thus significantly reduced memory copy overhead [87]. All these methods cannot effectively detect attacks based on encrypted traffic. For task-specific methods, Nelms et al. [80], Invernizzi et al. [81], and Bilge et al. [18] detected traffic in the different stages of malware campaigns by using statistical ML. Bartos et al. [88] and Tang et al. [37] detect malformed HTTP request traffic.

Task-Specific Encrypted Traffic Detection. The existing encrypted traffic detection relies on domain knowledge for short-term flow-level features [1], [2], [3]. For example, Zheng et al. leveraged SDN to achieve crossfire attack detection [16], and Xing et al. designed the primitives for the programmable switches to detect link flooding attacks [89]. For encrypted malware traffic, Bilge et al. [18] leveraged the traffic history to detect C&C server, and Tegeler et al. developed supervised learning using time-scale flow features extracted from malware binaries [6]. Anderson et al. studies the feasibility of detecting malware encrypted communication via malformed TLS headers [7]. To the best of our knowledge, our HyperVision is the first system that enables unsupervised detection for the encrypted traffic with unknown patterns.

Encrypted Traffic Classification. HyperVision aims to identify the malicious behaviors according to encrypted traffic. It is different from encrypted traffic classifications that decide if the traffic is generated by certain applications or users [90], [91], [92], [93]. For instance, Rimmer et al. leveraged DL for web fingerprint, which de-anonymizes Tor traffic by classifying encrypted web traffic [94]. Siby et al. showed that classifying encrypted DNS traffic can jeopardize the user privacy [31]. Similarly, Bahramali et al. classified the encrypted traffic of instant messaging applications [95]. Ede et al. designed semi-supervised learning for mobile applications fingerprinting [96]. All these classifications are orthogonal to HyperVision.

X. CONCLUSION

In this paper, we present HyperVision, an ML based realtime detection system for encrypted malicious traffic with unknown patterns. HyperVision utilizes a graph to represent flow interaction patterns, while not requiring prior knowledge on the traffic. Specifically, HyperVision uses two different strategies to represent the interaction patterns generated by short and long flows and aggregates the information of these flows. We develop an unsupervised graph learning method to detect the traffic by utilizing the connectivity, sparsity, and statistical features in the graph. Moreover, we establish an information theory based analysis framework to demonstrate that HyperVision preserves near-optimal information of flows for effective detection. The experiments with 92 real-world attack traffic datasets demonstrate that HyperVision achieves at least 0.86 F1 and 0.92 AUC with 0.29s detection latency.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their insightful comments.

REFERENCES

- [1] Cisco. *Cisco Encrypted Traffic Analytics*. Accessed: May 2022. [Online]. Available: <https://www.cisco.com/c/en/us/solutionsenterprise-networks/enterprise-network-security/eta.html>
- [2] E. Papadogiannaki and S. Ioannidis, "A survey on encrypted network traffic analysis applications, techniques, and countermeasures," *ACM Comput. Surv.*, vol. 54, no. 6, p. 123, Jul. 2021.
- [3] B. Anderson and D. McGrew, "Identifying encrypted malware traffic with contextual flow data," in *Proc. ACM Workshop Artif. Intell. Secur.*, Oct. 2016, pp. 35–46.
- [4] G. Pellegrino, M. Johns, S. Koch, M. Backes, and C. Rossow, "Deemon: Detecting CSRF with dynamic analysis and property graphs," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1757–1771.
- [5] K. Thomas et al., "Data breaches, phishing, or malware: Understanding the risks of stolen credentials," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1421–1434.
- [6] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, "BotFinder: Finding bots in network traffic without deep packet inspection," in *Proc. 8th Int. Conf. Emerg. Netw. Experiments Technol.*, Dec. 2012, pp. 349–360.
- [7] B. Anderson and D. A. McGrew, "Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity," in *Proc. SIGKDD*, 2017, pp. 1723–1732.
- [8] M. Javed and V. Paxson, "Detecting stealthy, distributed SSH brute-forcing," in *Proc. CCS*, 2013, pp. 85–96.
- [9] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting botnet command and control channels in network traffic," in *Proc. NDSS*, 2008.
- [10] M. S. Kang, S. B. Lee, and V. D. Gligor, "The crossfire attack," in *Proc. IEEE Symp. Secur. Privacy*, May 2013, pp. 127–141.
- [11] Z. Durumeric, M. Bailey, and J. A. Halderman, "An internet-wide view of internet-wide scanning," in *Proc. 23rd USENIX Secur. Symp.*, 2014, pp. 65–78.
- [12] C. Lever, P. Kotzias, D. Balzarotti, J. Caballero, and M. Antonakakis, "A lustrum of malware network communication: Evolution and insights," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 788–804.
- [13] B. Eshete and V. N. Venkatakishnan, "DynaMiner: Leveraging offline infection analytics for on-the-wire malware detection," in *Proc. 47th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2017, pp. 463–474.
- [14] X. Feng, C. Fu, Q. Li, K. Sun, and K. Xu, "Off-path TCP exploits of the mixed IPID assignment," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 1323–1335.
- [15] A. Oest et al., "Sunrise to sunset: Analyzing the end-to-end life cycle and effectiveness of phishing attacks at scale," in *Proc. 29th USENIX Security Symp.*, 2020, pp. 2039–2056.
- [16] J. Zheng, Q. Li, G. Gu, J. Cao, D. K. Yau, and J. Wu, "Realtime DDOS defense using COTS SDN switches via adaptive correlation analysis," *IEEE Trans. Inf. Forensics Security*, vol. 13, pp. 1838–1853, 2018.
- [17] M. Du, F. Li, G. Zheng, and V. Srikanth, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1285–1298.
- [18] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: Detecting botnet command and control servers through large-scale NetFlow analysis," in *Proc. 28th Annu. Comput. Secur. Appl. Conf.*, Dec. 2012, pp. 129–138.
- [19] Y. Mirsky, T. Doitsman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–15.
- [20] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. V. Ramos, and A. Madeira, "FlowLens: Enabling efficient flow classification for ML-based network security applications," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–18.
- [21] Cisco. *Cisco Systems NetFlow Services Export Version 9*, document RFC 3954, May 2022, doi: [10.17487/RFC3954](https://doi.org/10.17487/RFC3954).
- [22] Zeek. *An Open Source Network Security Monitoring Tool*. Accessed May 2022. [Online]. Available: <https://zeek.org/>
- [23] R. Zamir, "A proof of the Fisher information inequality via a data processing argument," *IEEE Trans. Inf. Theory*, vol. 44, no. 3, pp. 1246–1250, May 1998.
- [24] Intel. *Data Plane Development Kit*. Accessed: May 2022. [Online]. Available: <https://www.dpdk.org/>
- [25] H. L. J. Bijmans, T. M. Booij, and C. Doerr, "Just the tip of the iceberg: Internet-scale exploitation of routers for cryptojacking," in *Proc. CCS*, 2019, pp. 449–464.
- [26] Cisco. *Cisco SPAN*. Accessed: May 2022. [Online]. Available: <https://www.cisco.com/c/en/us/support/docs/swit-ches/catalyst-6500-series-switches/10570-41.html>
- [27] M. Zhang et al., "Poseidon: Mitigating volumetric DDoS attacks with programmable switches," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–18.
- [28] Q. Li et al., "Dynamic network security function enforcement via joint flow and function scheduling," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 486–499, 2022.
- [29] Q. Li, Y. Liu, Z. Liu, P. Zhang, and C. Pang, "Efficient forwarding anomaly detection in software-defined networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 11, pp. 2676–2690, Nov. 2021.
- [30] D. Rupperecht, K. Kohls, T. Holz, and C. Popper, "Call me maybe: Eavesdropping encrypted LTE calls with ReVoLTE," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 73–88.
- [31] S. Siby, M. Juarez, C. Diaz, N. Vallina-Rodriguez, and C. Troncoso, "Encrypted DNS \implies privacy? A traffic analysis perspective," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–19.
- [32] M. Jonker, A. King, J. Krupp, C. Rossow, A. Sperotto, and A. Dainotti, "Millions of targets under attack: A macroscopic characterization of the DoS ecosystem," in *Proc. Internet Meas. Conf.*, Nov. 2017, pp. 100–113.
- [33] R. Xie et al., "Disrupting the SDN control channel via shared links: Attacks and countermeasures," *IEEE/ACM Trans. Netw.*, vol. 30, no. 5, pp. 2158–2172, Oct. 2022.
- [34] J. Cao, M. Xu, Q. Li, K. Sun, and Y. Yang, "The LOFT attack: Overflowing SDN flow tables at a low rate," *IEEE/ACM Trans. Netw.*, vol. 31, no. 3, pp. 1416–1431, Jun. 2023.
- [35] X. Feng et al., "PMTUD is not panacea: Revisiting IP fragmentation attacks against TCP," in *Proc. NDSS*, 2022, pp. 24–28.
- [36] X. Feng et al., "Off-path network traffic manipulation via revitalized ICMP redirect attacks," in *Proc. 31st USENIX Secur. Symp.*, 2022, pp. 2619–2636.
- [37] R. Tang et al., "ZeroWall: Detecting zero-day web attacks through encoder–decoder recurrent neural networks," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Jul. 2020, pp. 2479–2488.
- [38] C. Fu, Q. Li, M. Shen, and K. Xu, "Realtime robust malicious traffic detection via frequency domain analysis," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 3431–3446.
- [39] IETF. *Specification of the IP Flow Information Export (IPFIX) Protocol*, document RFC 7011, Accessed: May 2022. [Online]. Available: <https://www.rfc-editor.org/info/rfc7011>
- [40] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Trans. Comput. Syst.*, vol. 21, no. 3, pp. 270–313, 2003.
- [41] WIDE. *MAWI Working Group Traffic Archive*. Accessed: May 2022. [Online]. Available: <http://mawi.wide.ad.jp/mawi/>
- [42] Z. Zhao, H. Sadok, N. Atre, J. C. Hoe, V. Sekar, and J. Sherry, "Achieving 100 Gbps intrusion prevention on a single server," in *Proc. OSDI*, 2020, pp. 1083–1100.
- [43] Z. Qian and Z. M. Mao, "Off-path TCP sequence number inference attack—How firewall middleboxes reduce security," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 347–361.
- [44] Z. Wang et al., "SymTCP: Eluding stateful deep packet inspection with automated discrepancy discovery," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–17.
- [45] J. Zeng, Z. L. Chua, Y. Chen, K. Ji, Z. Liang, and J. Mao, "WATSON: Abstracting behaviors from audit logs via aggregation of contextual semantics," in *Proc. NDSS*, 2021, pp. 1–18.
- [46] R. Yang, S. Ma, H. Xu, X. Zhang, and Y. Chen, "UISCOPE: Accurate, instrumentation-free, and visible attack investigation for GUI applications," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–18.
- [47] Cisco. *Network as a Security Sensor Threat Defense With Full NetFlow*. Accessed: May 2022. [Online]. Available: <https://www.cisco.com/c/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/white-paper-c11-736595.pdf>
- [48] J. Gan and Y. Tao, "DBSCAN revisited: Mis-claim, un-fixability, and approximation," in *Proc. SIGMOD*, 2015, pp. 519–530.
- [49] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified NP-complete problems," in *Proc. 6th Annu. ACM Symp. Theory Comput.*, 1974, pp. 47–63.
- [50] Microsoft. *A Theorem Prover From Microsoft Research*. Accessed: May 2022. [Online]. Available: <https://github.com/Z3Prover/z3>
- [51] Z. Liu et al., "Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric DDoS attacks with programmable switches," in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 3829–3846.

- [52] PcapPlusPlus. *A C++ Library for Capturing, Parsing and Crafting of Network Packets*. Accessed: May 2022. [Online]. Available: <https://pcapplusplus.github.io/>
- [53] Mlpack. *Mlpack: Open Source Machine Learning Library*. Accessed: May 2022. [Online]. Available: <https://www.mlpack.org/>
- [54] P. Richter and A. Berger, "Scanning the scanners: Sensing the internet from a massively distributed network telescope," in *Proc. Internet Meas. Conf.*, Oct. 2019, pp. 144–157.
- [55] D. Kopp, M. Wichtlhuber, I. Poese, J. Santanna, O. Hohlfeld, and C. Dietzel, "DDoS hide & seek: On the effectiveness of a booter services takedown," in *Proc. Internet Meas. Conf.*, Oct. 2019, pp. 65–72.
- [56] R. Miao, R. Potharaju, M. Yu, and N. Jain, "The dark menace: Characterizing network-based attacks in the cloud," in *Proc. Internet Meas. Conf.*, Oct. 2015, pp. 169–182.
- [57] Kaspersky. *Kaspersky Security Bulletin 2020. Statistics*. Accessed: May 2022. [Online]. Available: https://go.kaspersky.com/rs/802-IJN-240/images/KSB_statistics_2020_en.pdf
- [58] A. Kuzmanovic and E. W. Knightly, "Low-rate TCP-targeted denial of service attacks: The shrew vs. the mice and elephants," in *Proc. Conf. Appl., Technol., Architectures, Protocols Comput. Commun.*, Aug. 2003, pp. 75–86.
- [59] Y. Cao, Z. Qian, Z. Wang, T. Dao, S. V. Krishnamurthy, and L. M. Marvel, "Off-path TCP exploits: Global rate limit considered dangerous," in *Proc. 25th USENIX Secur. Symp.*, 2016, pp. 209–225.
- [60] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast internet-wide scanning and its security applications," in *Proc. USENIX Secur.*, 2013, pp. 605–620.
- [61] X. Luo and R. K. C. Chang, "On a new class of pulsing denial-of-service attacks and the defense," in *Proc. NDSS*, 2005, pp. 1–19.
- [62] X. Feng, Q. Li, K. Sun, C. Fu, and K. Xu, "Off-path TCP hijacking attacks via the side channel of downgraded IPID," *IEEE/ACM Trans. Netw.*, vol. 30, no. 1, pp. 409–422, Feb. 2022.
- [63] E. Liu, A. Nakanishi, M. Golla, D. Cash, and B. Ur, "Reasoning analytically about password-cracking software," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 380–397.
- [64] R. Merget et al., "Scalable scanning and automatic classification of TLS padding Oracle vulnerabilities," in *Proc. 28th USENIX Secur. Symp.*, 2019, pp. 1029–1046.
- [65] J. Caballero, C. Grier, C. Kreibich, and V. Paxson, "Measuring payer-install: The commoditization of malware distribution," in *Proc. 20th USENIX Secur. Symp.*, 2011, pp. 1–16.
- [66] H-Electric. *Internet Backbone and Colocation Provider*. Accessed: May 2022. [Online]. Available: <http://he.net/>
- [67] R. NCC. *The RIPE NCC is Building the Largest Internet Measurement Network Ever Made*. Accessed: May 2022. [Online]. Available: <https://atlas.ripe.net/>
- [68] CIC. *Canadian Institute for Cybersecurity Datasets*. Accessed: May 2022. [Online]. Available: <https://www.unb.ca/cic/datasets/index.html>
- [69] Stratosphere. *Real Malware Traffic Captures*. Accessed: May 2022. [Online]. Available: <https://www.strato-sphereips.org/datasets-overview>
- [70] A. Nappa, Z. Xu, M. Z. Rafique, J. Caballero, and G. Gu, "CyberProbe: Towards internet-scale active detection of malicious servers," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014.
- [71] G. Li et al., "Enabling performant, flexible and cost-efficient DDoS defense with programmable switches," *IEEE/ACM Trans. Netw.*, vol. 29, no. 4, pp. 1509–1526, Aug. 2021.
- [72] CIC. *DDoS Evaluation Datasets (CIC-DDoS2019)*. Accessed: May 2022. [Online]. Available: <https://www.unb.ca/cic/datasets/ddos-2019.html>
- [73] CIC. *Intrusion Detection Evaluation Datasets (CIC-IDS2017)*. Accessed: May 2022. [Online]. Available: <https://www.unb.ca/cic/datasets/ids-2017.html>
- [74] C. Fu, Q. Li, M. Shen, and K. Xu, "Frequency domain feature based robust malicious traffic detection," *IEEE/ACM Trans. Netw.*, vol. 31, no. 1, pp. 452–467, Feb. 2023.
- [75] S. Zhu et al., "You do (not) belong here: Detecting DPI evasion attacks with context learning," in *Proc. 16th Int. Conf. Emerg. Netw. Experiments Technol.*, Nov. 2020, pp. 183–197.
- [76] J. Holland, P. Schmitt, N. Feamster, and P. Mittal, "New directions in automated traffic analysis," in *Proc. CCS*, 2021, pp. 3366–3383.
- [77] Suricata. *An Open Source Threat Detection Engine*. Accessed: May 2022. [Online]. Available: <https://suricata-ids.org/>
- [78] C. Fu, Q. Li, and K. Xu, "Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2023, pp. 1–18.
- [79] B. J. Kwon, J. Mondal, J. Jang, L. Bilge, and T. Dumitras, "The dropper effect: Insights into malware distribution with downloader graph analytics," in *Proc. CCS*, 2015, pp. 1118–1129.
- [80] T. Nelms, R. Perdisci, M. Antonakakis, and M. Ahamad, "WebWitness: Investigating, categorizing, and mitigating malware download paths," in *Proc. 24th USENIX Secur. Symp.*, 2015, pp. 1025–1040.
- [81] L. Invernizzi et al., "Nazca: Detecting malware distribution in large-scale networks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 23–26.
- [82] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–58, Jul. 2009.
- [83] C. Fu, Q. Li, K. Xu, and J. Wu, "Point cloud analysis for ML-based malicious traffic detection: Reducing majorities of false positive alarms," in *Proc. CCS*, 2023, pp. 1005–1019.
- [84] Y. Qing et al., "Low-quality training data only? A robust framework for detecting encrypted malicious network traffic," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2024, pp. 1–18.
- [85] P. Li et al., "Learning from limited heterogeneous training data: Meta-learning for unsupervised zero-day web attack detection across web domains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2023, pp. 1020–1034.
- [86] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *Proc. 32nd USENIX Secur. Symp.*, 2023, pp. 1–18.
- [87] G. Siracusano et al., "Re-architecting traffic analysis with neural network interface cards," in *Proc. 19th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2022, pp. 513–533.
- [88] K. Bartos, M. Sofka, and V. Franc, "Optimized invariant representation of network traffic for detecting unseen malware variants," in *Proc. 25th USENIX Secur. Symp.*, 2016, pp. 807–822.
- [89] J. Xing, W. Wu, and A. Chen, "Ripple: A programmable, decentralized link-flooding defense against adaptive adversaries," in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 3865–3880.
- [90] M. Shen, J. Zhang, L. Zhu, K. Xu, and X. Du, "Accurate decentralized application identification via encrypted traffic analysis using graph neural networks," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 2367–2380, 2021.
- [91] M. Shen, J. Zhang, L. Zhu, K. Xu, and X. Du, "Subverting website fingerprinting defenses with robust traffic representation," in *Proc. 32nd USENIX Secur. Symp.*, 2023, pp. 607–624.
- [92] Q. Yin et al., "An automated multi-tab website fingerprinting attack," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 6, pp. 3656–3670, Nov. 2022.
- [93] X. Deng et al., "Robust multi-tab website fingerprinting attacks in the wild," in *Proc. IEEE Symp. Secur. Privacy*, May 2023, pp. 1005–1022.
- [94] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," in *Proc. NDSS*, 2018, pp. 1–15.
- [95] A. Bahramali, R. Soltani, A. Houmansadr, D. Goeckel, and D. Towsley, "Practical traffic analysis attacks on secure messaging applications," in *Proc. NDSS*, 2020, pp. 1–18.
- [96] T. Van Ede et al., "Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic," in *Proc. NDSS*, 2020, pp. 1–18.

Chuanpu Fu is currently pursuing the Ph.D. degree with Tsinghua University. His research interests include machine learning for security and network security.

Qi Li (Senior Member, IEEE) received the Ph.D. degree from Tsinghua University. He is currently an Associate Professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include internet and cloud security, mobile security, and big data security. He is an Editorial Board Member of IEEE TRANSACTIONS ON DEPENDABLE AND SECURITY COMPUTING.

Ke Xu (Fellow, IEEE) received the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University. He is currently a Full Professor with Tsinghua University. His research interests include next-generation internet, blockchain systems, the Internet of Things, and network security. He serves as the Steering Committee Chair for IEEE/ACM IWQoS and he has guest-edited several special issues for IEEE and Springer journals. He is a member of ACM.