OMAD5G: Online Malware Detection in 5G Networks using Compound Paths

Zhixin Wen
Binghamton University
School of Computing
Binghamton, NY, USA
zwen7@binghamton.edu

Guanhua Yan
Binghamton University
School of Computing
Binghamton, NY, USA
ghyan@binghamton.edu

Abstract

The pervasive growth of mobile devices has made them lucrative targets for various malware attacks. Despite numerous previous efforts on traffic-based malware detection, none of them offer a holistic approach to tackle the three challenges faced in online malware detection in 5G networks: *accuracy* (malware attacks can be detected at high accuracy), *scalability* (malware detection can be performed in a scalable manner for a large number of mobile users), and *integrality* (the malware defense system can be seamlessly integrated into the 5G network architecture). We thus propose a new system called **O**nline **MAl**ware **D**etection in **5G** Networks (OMAD5G) to defend against malware attacks in 5G networks. OMAD5G combines on-path malware detection, which is deployed by 5G User Plane Function usually handling large volumes of user traffic, and off-path malware detection, which is executed by separate servers with ample computational resources. We develop a lightweight classifier trained on Internet domain names for on-path malware detection, which is executed on P4 programmable switches, and apply powerful transformer neural networks on various packet features for off-path malware detection. OMAD5G uses tri-threshold learning based on the Neyman-Pearson criterion to facilitate toggling between on-path and off-path malware detection. The functionalities of OMAD5G are achieved with three 5G service-based architecture (SBA)-compliant call flows to enable easy integration into existing 5G networks. We implement a prototype of OMAD5G and evaluate its performances using two realistic mobile traffic datasets induced by human operations. Our results show that OMAD5G achieves a detection rate of ~95% and a false positive rate of ~1% for both datasets, and its on-path detection module outperforms NetBeacon, a state-of-the-art in-network traffic classifier deployable on P4 programmable switches, on both detection accuracy and deployment scalability.

CCS Concepts

• **Security and privacy → Mobile and wireless security**.

Keywords

5G networks, online malware detection, machine learning, P4 programmable switch

ACM Reference Format:
Zhixin Wen and Guanhua Yan. 2025. OMAD5G: Online Malware Detection in 5G Networks using Compound Paths. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '25), August 25–29, 2025, Hanoi, Vietnam.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3708821.3733864

1 Introduction

Proliferation of mobile devices has changed the landscape of Internet traffic drastically in the past decade. Over 56.8% of website traffic now originates from mobile devices, an increase of 75% from 2015 [45]. The new 5G technology will continue to drive the growth of mobile usage. According to GSMA Intelligence, there will be over two billion 5G connections worldwide by 2025 [44]. The pervasive growth of mobile devices has made them lucrative targets for various malware attacks such as adware, trojans, and ransomware.

Among the plethora of research efforts dedicated to mobile malware detection [36, 65], many focused on offline malware detection, which aims to detect malicious mobile apps through static analysis (e.g., RiskRanker [43], FlowDroid [16], Drebin [15], Apposcopy [38], and MAST [23]) and/or dynamic analysis in emulated environments (e.g., Andromaly [69], CopperDroid [72], AppsPlayground [66], TaintDroid [32], and VetDroid [83]). For online mobile malware detection, previous works applied traffic statistics-based methods [8, 14, 22, 25, 26, 70] and deep packet inspection (DPI)-based techniques [17, 25, 34, 37] to detect mobile malware traffic.

Despite these efforts, there has been lack of research on exploring mobile malware detection deployed as an integral service by mobile networks. Sitting between the subscribers' mobile devices and the wider Internet, mobile networks are at a unique position to safeguard these devices from malware attacks originating from the Internet. Mobile malware detection can be offered by a mobile network as a value-added service to its customers. Moreover, when a large army of mobile devices are infected by malware to create a mobile botnet, they can be used to harm or disrupt the operations of mobile networks [19, 33, 35, 54, 74, 75], which further provides incentives for mobile network operators to deploy effective malware defense mechanisms in their networks.

Online malware detection for 5G networks, needs to tackle the following challenges: malware attacks should be detected with high accuracy based on only mobile network traffic (*accuracy*), malware detection services can be offered to the mobile subscribers in a scalable fashion (*scalability*), and the malware detection service can be seamlessly integrated into the Service-Based Architecture (SBA) of a 5G network to notify the users whose mobile devices have been infected by malware (*integrality*). Although numerous methods have been proposed for traffic-based malware detection [11, 20, 37, 41, 48, 60, 79, 80, 84], none of them have provided a holistic approach that can overcome all these challenges (§ 3).

Against this backdrop, this work proposes a new system called **O**nline **MA**lware **D**etection in **5G** Networks (OMAD5G) to detect malware-infected mobile devices in 5G networks. OMAD5G applies a compound approach that integrates *on-path* and *off-path* malware detection to achieve both high deployment scalability and high detection accuracy. Its on-path malware detection module, which is deployed on the same paths traversed by user traffic in 5G networks, applies lightweight machine learning models on observed Domain Name Service (DNS) packets, which comprise only a small fraction of Internet traffic (e.g., 0.1% and 0.2% of IPv4-based Internet traffic in 2019 and 2020, respectively [67]) for quick malware detection. To overcome the limited discriminatory power of DNS name features for malware detection, the on-path malware detection module adopts two thresholds learned based on Neyman-Pearson criteria, one for low false positive rates and the other for low false negative rates. The first threshold is used as the first line of malware detection due to few false alarms raised to the mobile subscribers, while the second one is used to mirror the traffic of a small portion of mobile devices to separate servers for off-path malware detection. The off-path malware detection module, which is deployed at places to which user traffic in 5G networks are mirrored from their original paths and can access ample computational resources, applies powerful transformer neural networks [76] on various traffic features, such as flow characteristics, packet contents, and packet arrival timestamps, to achieve high detection accuracy. In order for seamless integration into 5G networks, OMAD5G introduces three 5G SBA-compliant call flows to facilitate provision and deployments of malware detection models within their data planes.

In a nutshell, our main contributions are summarized as follows. ① We propose a compound detection framework that leverages both on-path malware detection at line speed and off-path malware detection with high accuracy (§ 4). ② We develop lightweight malware detection models trained on Internet domain names for on-path malware detection and transformer neural networks trained on various features extracted from mobile traffic, such as flow characteristics, packet contents, and packet arrival timestamps, for off-path malware detection. We design a novel tri-threshold learning scheme based on the Neyman-Pearson criterion to search for joint threshold hyper-parameters that maximize detection rates while ensuring that the false alarm rates are below a pre-defined value (§ 5). ③ We demonstrate the feasibility of deploying the on-path malware detection module on commodity P4 programmable switches and the off-path malware detection module on separate computer servers. We design mechanisms to toggle between these two modules based on triggers derived from tri-threshold learning (§ 6). ④ We design three 5G SBA-compliant call flows to integrate key steps of OMAD5G service, namely, model deployment, detection enablement, and malware notification, into 5G networks seamlessly (§ 7). ⑤ We implement a prototype of OMAD5G based on an existing 5G network security testbed. We use two mobile traffic trace datasets induced by realistic human operations to evaluate the detection performances of OMAD5G, the resource usage of its on-path detection module on an Intel Tofino-based programmable switch, and its execution performances. Our experimental results show that OMAD5G achieves a detection rate of ∼95% and a false positive rate of ∼1% for both datasets, and its on-path detection module outperforms NetBeacon [84], a state-of-the-art in-network traffic classifier deployable on P4 programmable switches, on both detection accuracy and deployment scalability (§ 8). Our code is publicly available at: https://github.com/CyberSecurityScience/OMAD5G.

2 Related Work

Mobile malware traffic detection. Previous efforts on mobile malware traffic detection largely fall into two categories: shallow detection and deep packet inspections. Shallow detection methods only inspect packet headers such as port number, IP address and other traffic-based statistics while deep packet inspection (DPI) can examine the contents of the packets such as DNS queries and HTTP requests. One shallow detection method, which is widely used, is traffic statistics-based techniques [8, 14, 22, 25, 26, 70]. These works rely on traffic statistics features such as average packet sizes and flow durations for mobile malware detection. Traditional DPI-based intrusion detection systems such as Snort [2] and Zeek [3] apply pattern matching techniques to identify malicious network traffic, including those generated by malicious mobile apps. These systems usually perform poorly on malware detection when malware attacks constantly evolve with new variants. Recent methods have combined DPI with deep learning algorithms to improve detection accuracy of malicious mobile traffic [17, 25, 34, 37]. However, these previous approaches tend to be resource intensive and are thus not suitable for in-network malware detection in large-scale mobile networks. For instance, the traffic statistics features require collection and storage of various traffic attributes for a long period of time. Many of these traffic statistics are collected on a per flow basis, thus incurring significant overhead for online malware detection. The problem is further exacerbated when deep learning is used. To improve scalability, OMAD5G deploys a lightweight first-line defense directly on P4 programmable switches to detect malware at line speed for the majority of the traffic while mirroring only a small portion of traffic to a separate server for deeper examination.

Domain name-based malware detection. There have been numerous efforts on detecting malicious domains hosting malware [11, 12, 20, 27, 53, 60]. DNS traffic has been used to identify Domain Generation Algorithm (DGA)-based bot malware [13], Windows malware [61], and Android malware [59]. Although the on-path malware detector in OMAD5G also leverages DNS traffic to identify suspicious traffic, it is deployable on P4 programmable switches for malware detection at line speed.

Attack traffic detection on programmable switches. Several works considered DDoS attack traffic detection using programmable switches [51, 55, 82]. Chang *et al.* proposed an on-switch malware detector which uses the switch Application-Specific Integrated Circuit (ASIC) to extract features and the switch CPU to run the neural network model for malware detection [24]. Due to limited communication bandwidth between the switch CPU and ASIC (e.g., 32GB/s for PCIe 4.0 x16) and slow CPU inference, this solution cannot achieve malware detection at line speed (e.g., up to 6.4Tb/s for Intel Tofino-1 and 12.8Tb/s for Intel Tofino-2) in large 5G networks. A few recent works deployed machine learning models directly on the switch ASIC for malicious traffic detection on P4 programmable switches. Mousika uses binary decision trees for malware traffic detection based on flow statistics [79]. NetBeacon improves the

Table 1: Shortcomings of representative previous works for online malware detection in 5G networks

Category	Accuracy	Scalability	Integrality
PCAP-based [37, 41, 48]	✓	✗	✗
DNS-based [11, 20, 60]	✗	✓	✗
In-network [79, 80, 84]	✓	✗	✗

scalability by differentiating short and long flows and applying a multi-phase sequential model architecture [84]. Our experimental results have shown that the on-path detection module of OMAD5G outperforms NetBeacon on both detection accuracy and deployment scalability (see § 8.2).

5G network security. There have been a plethora of efforts on improving the security of emerging 5G networks [9, 18, 21, 29, 47, 49]. Our work has a different goal as it aims to address the accuracy, scalability, and integrality challenges in deploying online malware detection within 5G networks.

3 Background and Motivation

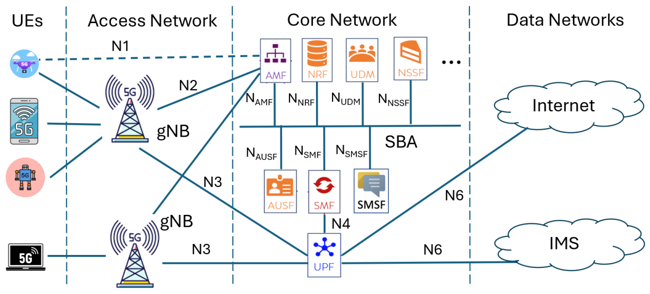


Figure 1: 5G network architecture. The labels shown on the connecting lines describes 5G communication interfaces.

A typical 5G network architecture is shown in Figure 1. A UE (User Equipment) is an end device (e.g., a mobile phone) that is capable of 5G communications. A 5G Radio Access Network (RAN) is comprised of one or more gNodeBs (gNBs), which are base stations using the New Radio (NR) technology. A 5G core network consists of its control plane following an SBA and its data plane. The key network functions (NFs) in the SBA include Access and Mobility Management Function (AMF), Session Management Function (SMF), Network Repository Function (NRF), Unified Data Management (UDM), Authentication Server Function (AUSF), Network Exposure Function (NEF), Network Slice Selection Function (NSSF), Policy Control Function (PCF), and Short Message Service Function (SMSF). The user plane is comprised of one or more User Plane Functions (UPFs), whose responsibilities include interconnecting 5G RANs and the external data networks (e.g., the Internet), packet routing/forwarding/inspection, Quality of Service (QoS) management, and usage reporting.

This work assumes a threat model of malicious mobile applications that run on mobile devices in a 5G network. As the traffic generated by these mobile apps needs to traverse the 5G network's user plane, the goal of this work is to investigate the feasibility of detecting malware-infected UEs based on their mobile network traffic. Towards this end, we need to tackle the following technical challenges: ① **Accuracy:** Can malware-infected UEs be detected in an online fashion based on their network traffic with high accuracy in a 5G network? ② **Scalability:** Can the online malware detection service be scaled up for a large number of mobile subscribers in a 5G network? ③ **Integrality:** Can the online malware detection service be integrated into the 5G network's SBA architecture to notify the users whose UEs have been infected by malware?

Despite many previous efforts on traffic-based mobile malware detection, none has offered a holistic approach to overcome all these challenges. Table 1 summarizes their shortcomings. For example, deep learning techniques have been applied on the features extracted from mobile traffic PCAP files to detect mobile malware with high accuracy [37, 41, 48]. Given the humongous amount of traffic in 5G networks, these methods lack the scalability to support online malware detection for a large number of mobile users. Researchers have proposed malware detection techniques based on DNS packets [11, 20, 60], which can be deployed close to DNS servers to monitor DNS request packets sent to suspicious domain names. Although they can achieve high scalability, their detection accuracy suffers due to the limited predictive power based on only DNS packets. Recently a few in-network traffic classification techniques have been developed to detect suspicious network traffic with impressive detection accuracy using flow statistics features on P4 programmable switches [24, 79, 84]. However, our experimental results (see § 8.2) have shown that they cannot be deployed efficiently along with UPFs on P4 programmable switches due to excessive computational resources needed to store the flow state features on the switches.

None of these previous works have considered integration into 5G networks for online malware detection. Although it is not hard to run these methods in the 5G data planes to detect malware traffic, it is difficult to attribute which mobile devices have been infected by malware. Traffic-based malware detection can identify the IP addresses from which malicious traffic originate, but linking these ephemeral IP addresses, which are dynamically assigned by SMFs, to the particular devices requires access to the internal states distributed at multiple NFs in the 5G control planes.

4 System Overview

To tackle the above challenges, we propose the OMAD5G system whose architecture is shown in Figure 2. OMAD5G can be offered as an opt-in value-added service by a mobile network to its subscribers for a particular data network (DN) (e.g., the Internet). Hence, whenever a UE requests to establish data connections to that DN, its service plan maintained by the 5G network is retrieved to check whether malware detection should be enabled for this device.

OMAD5G has three modules: *on-path malware detection*, *off-path malware detection*, and *model provisioning*. The on-path malware detection module runs along with the UPFs deployed in the data plane. As the UPFs have to deal with large volumes of mobile traffic in 5G networks, there may be limited computational resources available for on-path malware detection. Hence, the on-path malware detection module in OMAD5G applies a lightweight machine learning model D_{ON} on the Internet domain names requested by UEs to derive the likelihood that they have been infected by malware. For readers' convenience, we summarize the notations of the different machine learning models used by OMAD5G in Appendix A.

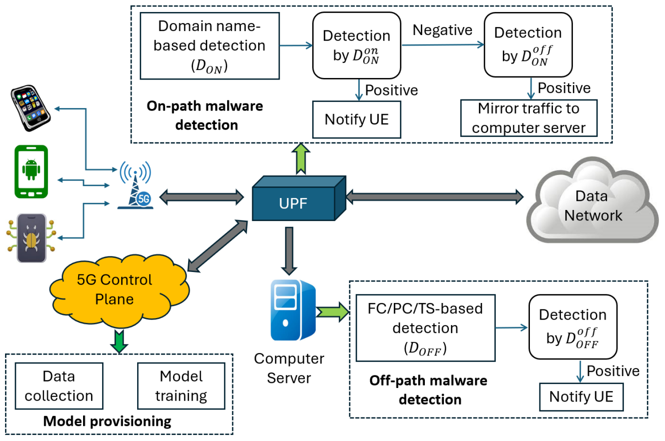

Domain name-based detection (D_{ON})
Detection by D_{ON}^{on}
Negative
Detection by D_{ON}^{off}
Positive
Positive
On-path malware detection
Notify UE
Mirror traffic to computer server
UPF
Data Network
5G Control Plane
Computer Server
FC/PC/TS-based detection (D_{OFF})
Detection by D_{OFF}^{off}
Positive
Off-path malware detection
Notify UE
Data collection
Model training
Model provisioning

Figure 2: The system architecture of OMAD5G

A unique design of OMAD5G is that the likelihood produced by model D_{ON} is compared against two separate thresholds to balance false positives and false negatives. The *low false positive (LFP)-threshold*, denoted by Θ_{ON}^{on}, allows quick malware detection based on only domain names with low false alarm rates. Using Θ_{ON}^{on}, it is however possible that the detector can miss some malware-infected UEs. Therefore, the *low false negative (LFN)-threshold*, denoted by Θ_{ON}^{off}, is used to achieve high detection rates, even at the sacrifice of false alarms. The alerts raised due to Θ_{ON}^{off} are *not* visible to the mobile users as they are only used to select a small portion of UEs whose traffic should be mirrored to the off-path detection module.

For ease of presentation, the detection model D_{ON} equipped with the LFP-threshold and LFN-threshold is called D_{ON}^{on} and D_{ON}^{off}, respectively. Based on these two detectors, the on-path malware detection module works as follows: if a UE's DNS traffic is detected to be positive by D_{ON}^{on}, a notification is sent to the UE. Otherwise, D_{ON}^{on} is further applied and a positive detection triggers its traffic to be mirrored to a separate computer server running off-path malware detection. Careful selection of both Θ_{ON}^{on} and Θ_{ON}^{off} enables infrequent false alarms raised by the on-path malware detection module while reducing the amount of traffic mirrored to the off-path malware detection module for further examination.

The off-path malware detection module applies a more powerful transformer neural network model, denoted by D_{OFF}, on a combination of flow characteristics (FC), packet content (PC), and packet arrival timestamp (TS) features for malware detection with high accuracy. Based on model D_{OFF}, a threshold hyperparameter, denoted by Θ_{OFF}^{off}, is used by the off-path malware detection module to balance its false positive and false negative errors. This malware detector using threshold Θ_{OFF}^{off} is called D_{OFF}^{off}. When the detection by D_{OFF}^{off} is positive, an alert is triggered to notify the infected UEs.

The compound scheme of combining both on-path and off-path malware detection can achieve both high detection accuracy and high scalability. Assuming that usually only a small fraction of mobile devices are infected by malware, the on-path malware detection module works as the first line of defense to achieve high scalability by filtering out the majority of mobile network traffic. Using the more advanced transformer neural networks on FC/PC/TS features, the complementary off-path malware detection module overcomes the limited discriminatory power of on-path malware detection based on only domain names to improve detection accuracy.

The prediction models needed for both on-path and off-path malware detection are trained by the model provisioning module. It is implemented by a new NF called User Safety Function (USF), which is responsible for ensuring the safety of users' data stored on their end devices. As part of the 5G SBA, USF can apply the same security mechanisms as other NFs for service authentication [5].

The model provisioning module uses labelled datasets to train the three detectors, D_{ON}^{on}, D_{ON}^{off}, and D_{OFF}^{off}. More specifically, it partitions these datasets into two portions: the *training* portion is used to train the two models D_{ON} and D_{OFF} while the *validation* portion is used for tri-threshold learning, which finds proper values for the three thresholds Θ_{ON}^{on}, Θ_{ON}^{off}, and Θ_{OFF}^{off}. In tri-threshold learning, we apply the Neyman-Pearson criterion [68, 73] to maximize the overall detection rate of OMAD5G while ensuring that its false positive rate should not exceed threshold ϕ within a targeted detection period.

Coordination among the three modules of the OMAD5G system is needed to achieve online malware detection in 5G networks. OMAD5G introduces three respective 5G SBA-compliant call flows to deploy the detection models trained by the USF in the 5G user plane, enable malware detection for a particular UE, and notify the UE after malware infection has been detected from its mobile traffic.

5 Model Provisioning

5.1 On-path detection model

OMAD5G uses the *Internet domain names* queried by mobile apps as features for malware detection. During the training phase, a reputation score is learned for each simplified domain name encountered in a DNS request packet. As there are numerous Fully Qualified Domain Names (FQDNs) on the Internet, making it difficult to learn a reputation score for each of them, we strip their prefix host names to obtain simplified domain names as follows. If a FQDN includes a gTLD (generic Top Level Domain), its simplified domain name includes its last two labels. For example, *www.eecs.mit.edu* and *math.mit.edu* share the same simplified domain names as *mit.edu*. Moreover, if a FQDN includes a ccTLD (Country Code Top Level Domain), we define its simplified domain name to include its three last labels. For example, the simplified domain name of *www.google.co.jp* is *google.co.jp*.

Let $\mathbf{M}$, where $|\mathbf{M}| = m$, denote the ordered set of all simplified domain names observed from the network traffic traces. Their reputation scores are learned from a labeled dataset $\mathbf{G}$ with $|\mathbf{G}| = n$. Each data point in $\mathbf{G}$ is a tuple $(\mathbf{X}_i, \mathbf{y}_i)$ abstracted from the network traffic trace of a particular mobile device, where $\mathbf{X}_i = \{c_i^{(j)}\}_{j=1,\dots,m}$ with $c_i^{(j)}$ giving the occurrence count of the j-th simplified domain name in set $\mathbf{M}$ (denoted by $\mathbf{M}^{(j)}$) within this trace while $\mathbf{y}_i \in \{0, 1\}$ indicates whether the trace contains malicious traffic or not.

Given dataset $\mathbf{G}$, the goal of reputation score learning is to learn a reputation score $r^{(j)} \in \mathbb{R}$ for each simplified domain name $\mathbf{M}^{(j)}$. We apply the mini-batch gradient descent approach to learn the reputation scores. We first define the aggregate reputation score learned for the i-th data sample as follows:

$$s^{(i)} = \sum_{1 \le j \le m} r^{(j)} \cdot c_i'^{(j)}, \tag{1}$$

where $c_i'^{(j)} = c_i{}^{(j)} + \delta$ is the augmented occurrence number of the j-th domain in set $\mathbf{M}$. We add a random jitter δ, where $\delta \sim \mathcal{N}(0, \kappa \cdot c_i{}^{(j)})$, to the occurrence count in *every* mini-batch gradient descent step with strength κ. Such jitters can improve the robustness of the model when it is deployed for inference as the occurrence counts of simplified domain names may fluctuate in the real world. By default we let κ be 0.5.

When applying the mini-batch gradient descent method, we use the weighted cross entropy loss as the objective function:

$$L = -\frac{1}{n} \sum_i \left\{ \lambda \mathbf{y}_i \log \sigma(-s^{(i)}) + (1 - \mathbf{y}_i) \log (1 - \sigma(-s^{(i)}) \right\}, \tag{2}$$

where $\sigma(\cdot)$ is the sigmoid function and λ can be used to help reduce false positive rates. By default we let λ be 0.5. The negative sign in front of $s^{(i)}$ ensures that a higher reputation score corresponds to a lower probability of being malicious.

The outcome of model training is a logistic regression model D_{ON}, which includes the reputation score for each simplified domain name. A mobile device usually contains system applications that constantly communicate with some system domains. As such system domain names give no indication of the device's infection state, we omit them from set $\mathbf{M}$. Appendix B lists some system domains used by Android.

During malware detection some domain names may not have been seen in the training dataset. We assign a special domain `UNKNOWN` to represent any unseen domain name. The reputation score of this `UNKNOWN` domain is also learned during the training phase. During each mini-batch gradient descent step and for each sample, we randomly choose a small fraction γ of the domain names and set them to be `UNKNOWN`. We let γ be 5% by default.

5.2 Off-path detection model

Off-path malware detection in OMAD5G uses a transformer neural network trained on flow characteristics, packet content, and packet arrival timestamp features. Each flow is uniquely identified by the standard 5-tuple, including the source IP address/port number, the destination IP address/port number, and the protocol in use. A flow is considered completed if any of the following is met: (1) a TCP flow with FIN flags sent in both directions; (2) a TCP flow with RST set in its packet; (3) a DNS response in a DNS flow with a corresponding request packet; and (4) any flow that has been inactive for 120 seconds or longer. For each flow we extract flow characteristics features such as number of bytes, ratio of uplink/downlink byte/packet counts, and mean packet inter-arrival time.

For packet content features, we consider the first N_{PC} bytes observed from each flow, including those from the IP headers. To prevent learning bias, we exclude the source or the destination IP address among these bytes. If a flow is completed with fewer than N_{PC} bytes, the remaining ones are padded with zeros.

Data preprocessing. Before feeding the raw features to the transfomer neural network, we preprocess them as follows. For each categorical feature, such as the transport or the application protocol name, we learn its projection into an $\mathbb{R}^4$ embedding space during the training phase. For each numerical feature x, we first perform normality test on x using the Shapiro–Wilk test [81] with the p-value set to 0.05. If the feature is normally distributed, we normalize each of its values, x_i, as $x_i' = \frac{x_i - \mu}{\sigma}$ where μ and σ are its mean and standard deviation, respectively. If X is not normally distributed, we apply the log transformation technique [31] to prevent failed model training due to Not a Number (NaN) loss or weights as follows: we first take the logarithm as $x_i'' = \log(1 + x_i - \min_j \{x_j\})$. Assuming that μ'' and σ'' are the mean and standard deviation of set $\{x_j''\}$, we obtain normalized values as $x_i' = \frac{x_i'' - \mu''}{\sigma''}$.

Model design. For each flow we use four types of projected features. First, its FC features are projected to 512-dimensional vector with a 3-layer Multilayer Perceptron (MLP) network with batch normalization and Gaussian Error Linear Unit (GELU) activation. Second, for PC features, the inputs are processed with a 3-layer convolutional network (CNN) to project the first N_{PC} bytes of a flow to another 512-dimensional vector. The first layer in the model has a kernel size of 20 and a stride of 10, the second one a kernel size of 15 and a stride of 1, and the last one a kernel size of 1 and a stride of 1. By default we use $N_{PC} = 160$ bytes to balance model complexity and detection accuracy. Third, we encode a flow's start and end timestamps into a 512-dimensional vector using a similar idea of sinusoidal positional encoding in [76]. Let $t \in \{0, 1800\}$ be a timestamp. We compute its d-dimensional encoding as a vector:

$$p_t^{(i)} = \begin{cases} \sin(w_k.t), & \text{if } i = 2k \\ \cos(w_k.t), & \text{if } i = 2k + 1 \end{cases} \tag{3}$$

where $w_k = 1/(10000^{2k/d})$. We set $d = 256$ so the two respective encodings for start and end timestamps of each flow can be concatenated to create a 512-dimensional vector.

The three 512-dimensional vectors are summed to form a single 512-dimensional feature vector for each flow. All flows' feature vectors, together with a learned `[CLS]` token, are fed into a two-layer transformer neural network [76]. The output of `[CLS]` token is used for malware prediction via another small MLP, denoted by MLP_{CLS}. The output from MLP_{CLS} is a score between $[0, 1]$ indicating the likelihood that the UE has been infected by malware.

Model training. To bound the model size so that it can fit into the GPUs, we randomly drop out flows for each UE to ensure that the number of flows per UE does not exceed a certain threshold, θ_f. By default we let θ_f be 4000. We train our model using the AdamW [56] optimizer with a batch size of 24 and a weight decay of 0.1 for 10 epochs. We use a learning rate of 3e-4 at the beginning and decrease it to 3e-5 and 3e-6 at epoch 5 and 8, respectively. The outcome of model training is called D_{OFF} as seen in Figure 2. Appendix C presents the experimental results about the effects of different feature types and model hyperparameters on the detection accuracy of the off-path detection model.

5.3 Tri-threshold learning

After the model training phase, we have obtained a logistic regression model D_{ON} for on-path malware detection and a transformer neural network D_{OFF} for off-path malware detection. Although these two models are trained independently using different features, when they are deployed, the execution of off-path malware detection hinges upon the result of on-path malware detection as seen from Figure 2. Such dependency renders it difficult to control the Type I errors (false positives) and Type II errors (false negatives) of the overall malware detection system. Therefore, we use a validation dataset to search joint optimal threshold hyperparameters so that the compound malware detector satisfies the Neyman-Pearson criterion [68, 73], which is to maximize the detection rate, or equivalently, to minimize the false negative rate, while ensuring that the false positive rate does not exceed a pre-defined tolerance level ϕ.

For on-path malware model D_{ON}, two thresholds are used: **(1)** ***LFP-threshold*** Θ_{ON}^{on}: if the aggregate reputation score is below Θ_{ON}^{on}, the UE is deemed to contain malicious traffic with high confidence and an alert is thus sent to the UE. **(2)** ***LFN-threshold*** Θ_{ON}^{off}**:** if the aggregate reputation score is between Θ_{ON}^{on} and Θ_{ON}^{off}, the UE is likely to contain malicious traffic and its traffic is mirrored to the off-path detection module for further examination.

For off-path malware detection model D_{OFF}, the score produced by MLP_{CLS} is compared against threshold Θ_{OFF}^{off} to decide whether the UE is infected by malware or not. The malware detector using threshold Θ_{OFF}^{off} is called D_{OFF}^{off}.

Due to dependency between D_{ON} and D_{OFF}, we use a validation dataset to simulate the detection process for a certain period of time, say, a day, and estimate the false positive and false negative rates of the compound malware detector within that period. We assume that both on-path and off-path malware detection modules are executed periodically every ω time units. We use the validation dataset to simulate the detection process for $k \cdot \omega$ time units. For each threshold triple $(\Theta_{ON}^{on}, \Theta_{ON}^{off}, \Theta_{OFF}^{off})$, we can apply the two malware detection models D_{ON} and D_{OFF} as shown in Figure 2 to simulate malware detection within the $k \cdot \omega$ time units and derive the overall false positive and false negative rates of the compound malware detector during this period. The tri-threshold learning phase aims to find such a threshold triple that leads to a false positive rate below the pre-defined threshold ϕ while minimizing the false negative rate.

From a practical perspective, we should put a limit on how much traffic should be mirrored to the off-path malware detection module, depending upon the amount of resources available to it. Therefore in our design we define a maximal fraction of UEs, denoted by η, whose traffic can be forwarded to the off-path detection module. During tri-threshold learning, if on average more than η of the UEs have to use off-path malware detection, the threshold triple is deemed as invalid. More specifically, consider k detection periods, each lasting ω time units. We assume that there are u UEs. Let q_i be the number of UEs for which off-path malware detection is enabled during the i-th period. For a given threshold triple, if $\frac{\sum_{i=1}^{k} q_i}{k \cdot u} \geq \eta$, this configuration is deemed likely to overload the off-path detection module and is thus treated as invalid.

6 Malware Detection

6.1 On-path malware detection

Currently, OMAD5G assumes that 5G UPFs are deployed on P4 programmable switches. P4 programmable switch-based UPFs usually adopt an architecture where its pre-queuing ingress stage performs traffic classification and packet routing/forwarding and its post-queuing egress stage does post-QoS accounting [57, 78].

The malware detection module deployed on a P4 programmable switch is illustrated in Figure 3. It consists of two components: a *controller* and a *monitor*. Although they are both located on the same switch device, the controller is executed by the switch CPU while the monitor runs on the switch ASIC. The controller keeps track of all active PDU sessions whose traffic should be examined for malware infection. A timer is created for each such session, which is periodically fired every ω time units to read aggregated reputation scores from the switch ASIC and compare it against thresholds Θ_{ON}^{on} and Θ_{ON}^{off} to decide whether the PDU session is infected by malware or whether its traffic should be mirrored to the off-path detection module. The monitor, which is implemented as P4 programs, parses DNS packets for each PDU session and aggregates the reputation scores of observed simplified domain names based on the on-path malware detection model (i.e., D_{ON}) pre-trained by the USF.

Feature extraction. To extract simplified domain names as features from DNS packets, we follow a previous approach [50] by extracting domain labels into chunks whose sizes are always the powers of two. The domain labels are reversed at the ingress deparsing step and reversed again at the egress deparsing step. As the labels in a domain name appear in a reverse order at the egress stage, the Top Level Domain and Second Level Domain labels can be easily identified to obtain the simplified domain name.

After domain name extraction the chunks are fed into the *domain suffix count table* to determinate the length of its suffix. The domain suffix count table is an exact-match table that stores the number of labels in a domain suffix. For example *www.google.com* has a single suffix, *com*, while *www.google.co.jp* has two suffix labels, *co.jp*. In our implementation at most two suffix labels are used for each FQDN. Based on the suffix length, we further extract an additional label from the domain name to construct the simplified domain name, which is the concatenation of this additional label and the suffix label(s). We next apply the CRC32 hash function provided by the programmable switch to obtain a 32-bit domain name key, which is used to identify a simplified domain name feature used by the classification model.

The hash value of a simplified domain name is used by the *domain reputation score table* to maintain the quantized reputation score learned for each simplified domain name. In this table the system domain names have reputation scores of 0 and therefore their appearances have no effects on malware detection. To support multiple classification models, each used for a different mobile ecosystem, the key to the domain reputation score table also includes a model identifier, which is associated with every PDU session whose detection service is enabled. To handle `UNKNOWN` domain names, a separate table is used to map a model identifier to a reputation score. For a particular model identifier, if the domain name hash does not match any entry in the domain reputation score table, the score is derived from this table instead.

Reputation score quantization. The reputation scores learned for simplified domain names are floating point numbers (see § 5.1), which cannot be directly used on programmable switches. We use the following scheme to quantize these floating point numbers into integers. We first map the entire range of possible scores into the range of 9-bit signed integer and store it in a 16-bit signed integer.

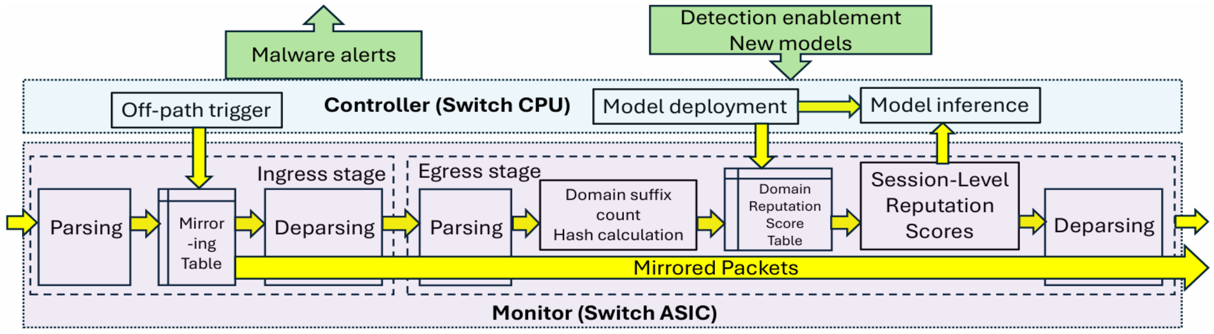


Figure 3: On-path malware detection deployed on a P4 programmable switch

Let $\widehat{r}^{(j)}$, where $1 \le j \le m$, be the quantized reputation score of the j-th simplified domain name $\mathbf{M}^{(j)}$ used during model inference. As represented by a 9-bit signed integer, we have $-256 \le \widehat{r}^{(j)} \le 255$. In model quantization, we store each 9-bit signed integer in a 16-bit register to prevent integer overflow during summation.

Suppose that *all* the reputation scores learned from the previous step fall into range $[l, h]$, where their values can be positive or negative. We define the quantization function:

$$Q(x) = \lfloor \frac{x}{\max\{|l|, |h|\}} \cdot (2^{9-1} - 1) \rceil, \tag{4}$$

where $\lfloor \cdot \rceil$ denotes the nearest integer. The quantized representation of reputation score $r^{(j)}$ is given by $Q(r^{(j)})$.

Model inference. OMAD5G performs online malware detection periodically for each PDU session whose detection service flag is enabled. It schedules an *on-path inference timer* every ω time units to invoke on-path malware detection. The switch ASIC keeps an exact-match table to store session-level aggregate reputation scores. An aggregate reputation score is stored as a 16-bit signed integer using a Register extern on the P4 programmable switch. For *each* DNS *query* packet with a simplified domain name d, its reputation score $\widehat{r}_d$ is retrieved from the domain reputation score table and next added to the existing aggregate reputation score of its PDU session s: $g_s = g_s \boxplus \widehat{r}_d$. $\boxplus$ denotes saturated addition which prevents overflow. We only consider DNS *query* packets because they demonstrate the intents of an app even under unsuccessful resolution.

When an aggregated reputation score g_s is retrieved from the switch ASIC for a PDU session, the controller compares it against threshold Θ^{on}_{ON} to determine whether it contains malware activities. If malware infection is detected, g_s is reset to 0 and a malware notification call flow is initiated. Otherwise, score g_s is compared against threshold Θ^{off}_{ON} to decide whether it is necessary to trigger the off-path malware detection module for further examination.

6.2 Off-path malware detection

The off-path malware detection module consists of three components: *off-path trigger*, which is executed by the controller of the P4 programmable switch as shown in Figure 3, *traffic collector*, which runs on a separate computer server with sufficient computational resources to process mirrored packets, and *off-path detector*, which has accesses to GPUs needed for malware detection based on the pre-trained transformer neural network model. When off-path detection is enabled for a PDU session, the off-path trigger inserts a new entry into the mirroring table stored inside the switch ASIC to mirror all its packets to the traffic collector and then suspends on-path malware detection for it. The off-path trigger also sends to the traffic collector a request message including the IP address of the UE whose traffic should be examined. Meanwhile, the off-path trigger schedules a *off-path inference timer* for the PDU session, which will be fired when an off-path detection period of ω time units ends for this session.

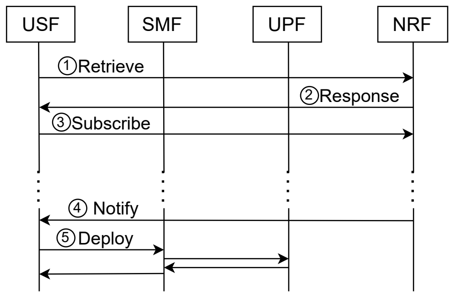
When the off-path inference timer fires, the off-path trigger sends a request message, which includes the UE's IP address and the traffic collector's URL (there may be multiple traffic collector instances running to achieve scalability), to the off-path detector. On the arrival of this message, the off-path detector queries the corresponding traffic collector to obtain the flow characteristics, packet content, and packet arrival timestamp features extracted for the UE. The off-path detector feeds all these features to the transformer neural network model shown in Figure 3. Using threshold Θ^{off}_{OFF}, the detection model makes the prediction and sends the detection result to the off-path trigger within an inference response message. If malware infection is detected, the same procedure is followed as in the on-path malware detection module to notify the UE. On-path detection resumes after off-path detection regardless of its detection result.

7 OMAD5G call flows

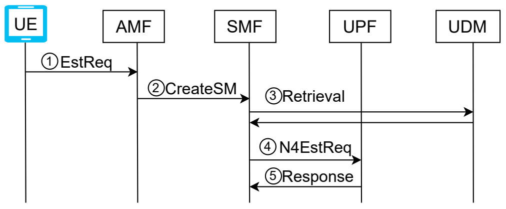
OMAD5G introduces three 5G SBA-compliant call flows to support online malware detection in a 5G network. The *model deployment* call flow deploys pre-trained malware classification models in the data plane of the 5G network. OMAD5G allows multiple models deployed in the same 5G network, each trained for a different mobile ecosystem (e.g., Android and iOS). The *detection enablement* call flow enables the data plane to know which PDU sessions should be monitored for malware infection. The *malware notification* call flow sends a short message notification to an infected UE, if its PDU sessions are detected to contain malware traffic.

Model deployment. The USF is responsible for training detection models D_{on} and D_{off}, based on labeled mobile network traffic traces, and finding the threshold triple $(\Theta^{on}_{ON}, \Theta^{off}_{ON}, \Theta^{off}_{OFF})$ using a validation dataset. Figure 4(1) presents the call flow to deploy both detection models and the threshold triple from the USF to the UPFs.

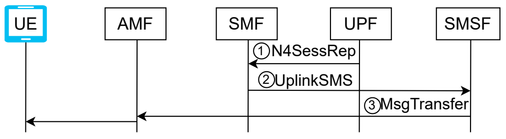
The call flow involves the following steps. ① Once started the USF queries NRF for SMF(s) with OMAD5G support. ② The NRF



(1) **Model deployment**



(2) **Detection enablement**



(3) **Malware notification**

Figure 4: Call flows used by OMAD5G

returns a list of online SMFs to the USF in its response. ③ The USF subscribes with the NRF to be notified when a new SMF comes online. ④ Once a new SMF becomes online, the NRF notifies the USF. ⑤ For each online SMF, the USF requests to deploy malware classification models on its associated UPFs. Note that Step ⑤ can happen right after Step ② for those SMFs on the NRF's response list. Between ④ and ⑤ a few steps used for authentication are omitted for simplicity.

Detection enablement. A UE can choose to enable OMAD5G for a specific DN (e.g., the Internet) as part of its service plan. The call flow to enable malware detection for a particular UE is illustrated in Figure 4(2). It involves the following steps. ① When the UE wants to connect to a DN, it sends a PDU Session Establishment Request to the AMF. ② The AMF processes this message and sends an Nsmf_PDUSession_CreateSMContext Request message to the SMF. ③ The SMF retrieves from the UDM the information needed to set up a connection (e.g., the maximum bandwidth). So far the call flow follows the standard PDU session establishment procedure [4]. To support malware detection, an extra flag is added to the retrieved data, indicating whether this subscriber requires the OMAD5G service for this DN. ④ If the service is enabled, when a PDU session is established for a given DN the SMF tells the PDU Session Anchor (PSA) UPF to enable malware detection in the N4 Session Establishment message, which is also used to create the context and install packet forwarding rules in the UPF. The PSA UPF serves as the anchor point for a PDU session to a given DN for a geographical region. As long as the UE stays in this region all its traffic goes through the PSA UPF before reaching the DN, which preserves the same IP address and connection. Therefore, installing malware classification models on the PSA UPF allows for persistent malware detection even under device mobility. ⑤ The UPF notifies the SMF that the PDU session has been successfully established for the UE in the data plane.

Malware notification. The call flow notifying an UE of malware infection, shown in Figure 4(3), includes three steps. ① Once a PDU session is detected to contain malware traffic, the UPF sends out an alert to the SMF using a N4 Session Report Request message [4]. ② The SMF requests the SMSF to send a text message to the UE about malware infection. This is done via the SMSF's Nsmsf_SMService_UplinkSMS API. ③ The message is sent to the UE via AMF using its Namf_Communication_N1N2MessageTransfer.

8 Evaluation

We have developed a prototype of the OMAD5G system based on VET5G [77], which is a testbed dedicated to 5G network security experimentation. We modify its SMF, NRF and UDM, which are implemented in Rust, to support the call flows introduced by OMAD5G. We implement the USF in Python due to its use of PyTorch for model training. We choose HiP4-UPF [78] as the 5G UPF implementation deployed on P4 programmable switches because its various optimization techniques improve scalability over other implementations [57, 63]. We make HiP4-UPF run on an Intel Tofino-based Netberg Auaro 710 programmable switch and implement the controller and the monitor as shown in Figure 3. The controller running on the switch CPU is implemented in Rust along with the existing UPF's controller program, while the monitor is implemented in the P4 language. The USF contains around 1400 lines of Python code and the modified UPF uses about 2,300 lines of P4 code. The traffic collector in the off-path detection module has around 1500 lines of Rust code and the off-path detector based on the transformer neural network has around 380 lines of Python code. More implementation details are provided in Appendix D.

8.1 Dataset description

Although there are many public mobile malware sample datasets, few provide mobile traffic traces induced by *realistic human operations*. The popular *CICAndMal2017* dataset [52] is the only one we have found that contains traffic traces collected from the executions of a large number of both malicious and benign Android apps. The dataset contains 426 malicious and 1700 benign pcap traces. To address the dataset scarcity issue, we created the *ZooBazaar2024* dataset by downloading the latest Android apps from both AndroZoo [10] and MalwareBazaar [7]. We asked forty students to run these applications within Android Emulator (Version 31.1.4.0) for at least five minutes to stimulate realistic mobile traffic. Each app was executed only once. Eventually, we obtained PCAP traces of 474 benign and 515 malicious apps for the ZooBazaar2024 dataset.

Dataset statistics. The basic statistics of both the CICAndMal2017 and ZooBazaar2024 datasets are given in Table 2. We notice that on average each PCAP trace in the CICAndMal2017 dataset lasts longer than that in the ZooBazaar2024 dataset, but there are fewer packets in each PCAP trace in the former dataset than the latter. Although the CICAndMal2017 dataset includes fewer DNS

Table 2: Basic statistics (means and standard deviations) of PCAP traces in CICAndMal2017 and ZooBazaar2024 datasets

Dataset	Label	Samples	Duration (seconds)	Packets	DNS Request Packets	Distinct Simplified Domain Names
CICAndMal2017	Benign	1700	636.4 ± 515.8	11327.7 ± 10788.9	129.1 ± 125.3	35.7 ± 16.0
	Malicious	426	2766.6 ± 527.7	55197.1 ± 214924.1	643.4 ± 337.4	110.7 ± 41.2
ZooBazaar2024	Benign	474	453.9 ± 393.6	66026.1 ± 99787.8	302.3 ± 196.8	27.1 ± 25.8
	Malicious	512	371.7 ± 413.5	59636.9 ± 84798.7	1130.2 ± 13615.1	79.6 ± 1106.9

request packets in each PCAP trace, its average number of distinct simplified domain names in each trace is higher than that in the ZooBazaar2024 dataset. Also, the domain names requested by the malware samples in the ZooBazaar2024 dataset exhibit high variation: there are 7 malware apps with more than 100 distinct simplified domain names, while 34 of them request fewer than 10 distinct simplified domain names.

The differences in traffic characteristics between the two datasets can result from multiple factors. First, the mobile apps used in the ZooBazaar2024 dataset are much more recent than those in the CICAndMal2017 dataset. Second, the mobile traffic traces were obtained from real smart phones for the CICAndMal2017 dataset, while for the ZooBazaar dataset they were captured from emulated Android devices operated by different human users. Using these two different datasets can provide more insights into the performances of OMAD5G when it is deployed in diverse real-world scenarios.

Data mixing. Many PCAP traces last only for a short period. As in practice mobile users may switch among different apps on their phones, we concatenate multiple PCAP traces to create a mixed dataset with longer periods of PCAP traces as follows. To support 5-fold nested cross validation, we first divide the PCAP files in each dataset into three types (i.e. training, validation, testing) and we only mix PCAP files from the same type. The training set is used to train the models, the validation set is used for tri-threshold learning, and the testing set is used for performance assessment.

The mixed traffic trace for each UE is created as follows. For one *not* infected by malware, we concatenate randomly chosen benign PCAP files until the total duration exceeds T time units. To create a sample traffic trace for a malware-infected UE, we mix benign and malicious pcaps from the same type by choosing α benign PCAP files after each malicious one. By default we let α, which is called a *mixing ratio*, be three. We keep selecting malicious and benign PCAP files with replacement until the total duration exceeds T time units. Thereafter we randomize the order of all selected PCAP files and concatenate these PCAP traces to simulate the UE's traffic. We name the mixed datasets based on CICAndMal2017 and ZooBazaar2024 as *CICAndMal2017-mixed* and *ZooBazaar2024-mixed*, respectively. After data mixing, the domain name statistics are explained in Appendix E.

8.2 Comparison with existing methods

PCAP-based detection. A number of methods have used the PCAP files in the CICAndMal2017 dataset to evaluate malware detection performances and they can be classified into two categories: traditional models [8, 52] and deep learning models [37, 41, 48]. We thus use it to justify the choices of both on-path and off-path malware detection models used by OMAD5G. For either model, we use a threshold of 1% false positive rate to search its hyperparameters based on the validation dataset. We measure malware detection performances using precision and recall metrics at the PCAP trace level. Given that the numbers of true positives, false positives, true negatives, and false negatives are TP, FP, TN, and FN, respectively, the precision is TP / (TP + FP) and the recall is TP / (TP + FN).

Table 3 compares the precisions and recalls of different methods based on the CICAndMal2017 dataset. The results show that the domain name-based on-path malware detector is comparable to the traditional models, although applying the 1% false position rate threshold in model training renders it to have much higher precisions while sacrificing recalls. We believe that this is a right choice as a malware detection system with high false alarm rates is usually less useful in practice [40, 71]. On the other hand, the off-path malware detector in OMAD5G has comparable performances as the existing deep learning-based methods.

Table 3: Performance comparison with previous malware detectors based on the CICAndMal2017 dataset. Deep learning models are indicated by *.

Work	Precision	Recall	Algorithm
Lashkari [52]	85.88%	88.30%	Random Forest
Lashkari [52]	85.40%	88.10%	KNN
Lashkari [52]	85.10%	88.00%	Decison Tree
Abuthawabeh [8]	86.65%	89%	Random Forest
Feng [37]	99.2%	98.2%	CACNN*
Imitiaz [48]	93.5%	93.4%	DeepAMD*
Gohari [41]	99.79%	99.5%	CNN-LSTM*
OMAD5G:D_{ON}	94.46%	82.93%	Section 5.1
OMAD5G:D_{OFF}	97.40%	99.76%	Section 5.2*

In-network detection. We perform experiments to compare the on-path malware detector D_{ON} in OMAD5G and a state-of-the-art in-network traffic classifier, NetBeacon [84]. We choose NetBeacon for performance comparison because its implementation code is publicly available [1], it has better detection accuracy and scalability than previous solutions [79], and its line-speed detection capability cannot be achieved by the previous work using the switch CPU for inference [24]. NetBeacon differentiates short and long flows. For short flows, it uses pure per-packet features to detect suspicious traffic so there is no need to store states for them. For long flows, it first uses hardware hashing to map them to a fixed number of stored states and then applies multi-phase sequential models on flow-level features for traffic detection. Decision tree forest models (e.g., Random Forest and XGBoost) are used to classify traffic flows.

The vanilla NetBeacon does not support integration with any existing 5G UPFs. We thus port it onto a P4 programmable switch

Table 4: Comparison of UPF functionalities and detection performances for on-path malware detection

Functionalities & Performances	NetBeacon -full	NetBeacon -partial	OMAD5G (D_{ON})
Simple PDR	✓	✓	✓
Complex PDR	✗	✗	✓
Forwarding	✗	✓	✓
IP routing	✗	✓	✓
Usage reporting	✗	✓	✓
QoS enforcement	✗	✗	✓
Number of UEs	59K	15K	124K
Precision	94.00%	93.63%	94.46%
Recall	62.82%	63.05%	82.93%

along with HiP4-UPF [78] to build a new UPF named as *NetBeacon-UPF*. As NetBeacon performs traffic classification based on network flows while OMAD5G detects malware traffic at the UE level, we let NetBeacon-UPF use the following rule for UE-level malware detection: for any UE, if more than ξ flows associated with it have been detected to be malicious by NetBeacon, the UE is deemed to be malware-infected. In our experiments, we search threshold ξ to achieve a targeted precision of 95%. NetBeacon-UPF also uses the default number (65,536) of stored states for hashed long flows.

Despite our best effort, we could not build a functional NetBeacon-UPF with all UPF features included. When the requested resources exceed the limit available on the P4 programmable switch, the P4 compiler outputs an error message. Hence, we consider two different approaches to reduce the resource usages. In the *NetBeacon-full* solution, we keep all NetBeacon functionalities intact but remove some UPF functionalities. We find that almost all UPF functionalities except traffic classification based on the simple PDR (Packet Detection Rule) table have to be removed. In the *NetBeacon-partial* solution, we remove two features used by NetBeacon, minimum packet size and variance of packet size, so only QoS enforcement and packet classification based on the complex PDR table have to be excluded from the UPF. By contrast, OMAD5G is able to work with all UPF functionalities enabled.

Table 4 summarizes the comparison results with NetBeacon. The NetBeacon-full and NetBeacon-partial approaches can support 59K and 15K UEs, respectively. Although NetBeacon-full uses all NetBeacon functionalities, it disables most of UPF functionalities, which allows it to accommodate more UEs than NetBeacon-partial. By contrast, OMAD5G can support as many as 124K UEs, suggesting its superior scalability. NetBeacon-full and NetBeacon-partial have similar detection performances with precision at ~94% and recall at ~63%. In comparison, the D_{ON} detector in OMAD5G can achieve a recall of 82.93%, which is a 31.6% improvement with a similar precision. These experimental results have demonstrated that *our DNS-based on-path malware detector outperforms NetBeacon on both detection accuracy and deployment scalability.*

8.3 Integrated detection performances

As shown in Figure 2, OMAD5G uses three separate malware detectors, two for on-path malware detection (D_{ON}^{on} and D_{ON}^{off}) and one for off-path malware detection (D_{OFF}^{off}). We use the CICAndMal2017-mixed and ZooBazaar-mixed datasets to evaluate the performances of integrated detection where all three malware detectors are used. As PCAP traces are concatenated, we need to consider how the detection windows decided by ω overlap with the original PCAP traces. It is possible that a detection window does not contain sufficient malicious traffic for malware detection. When we generate malicious training samples, we require a malicious training sample to be a detection window that either (a) covers at least a fraction β of an individual malicious PCAP trace or (b) has at least a fraction β of its duration overlapping with one or more malicious individual PCAP traces. We call fraction β the *overlapping ratio*.

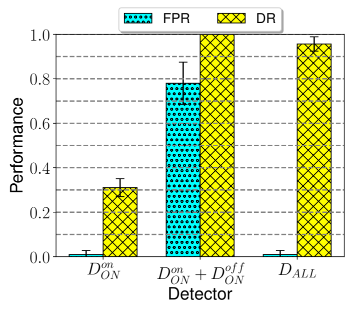
We let each detection window in both on-path and off-path malware detection last 30 minutes (i.e., ω = 30 minutes). To generate benign samples, we mix only benign individual PCAP traces. We then apply 30-minute windows on the concatenated traces to extract domain name features and FC/PC/TS ones for the benign samples. To create malicious samples, we mix both benign and malicious individual PCAP traces with a mixing ratio of $\alpha = 3$. From these concatenated PCAP traces, we use only those 30-minute windows with an overlapping ratio of at least $\beta = 0.7$ as malicious samples for model training. In total we have created 10,200 training windows, including 5,100 benign samples and 5,100 malicious ones.

For the validation dataset, we create 1,200 mixed traffic traces, each lasting at least four hours. They include traffic traces of 120 malware-infected UEs and 1,080 clean ones. Hence, the fraction of malware-infected UEs is 10%. Similarly, for testing purposes, we also create 1,200 mixed PCAP traces with the same fraction of malware-infected UEs. We measure the detection accuracy at the UE level: a clean UE alerted to be malware-infected is treated as a false positive. Otherwise, if a malware-infected UE is missed by the malware detector, it is deemed as a false negative. We choose false positive rate (FPR) and detection rate (DR), where FPR = FP / (FP + TN) and DR = TP / (TP + FN), for performance evaluation.

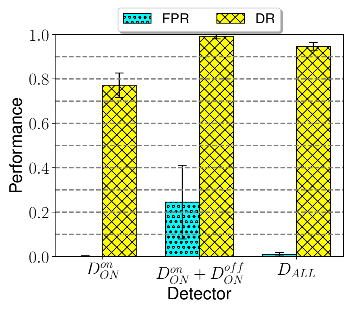
We set parameter η, which is used in tri-threshold learning as the limit on the fraction of UEs whose traffic can be mirrored for off-path malware detection, to be 40% by default. The targeted false positive rate (i.e., parameter ϕ) is set to be 1%.

As shown in Figure 2, the operations of the three detectors (i.e., D_{ON}^{on}, D_{ON}^{off}, and D_{OFF}^{off}) depend upon each other. Figure 5 compares detection performances of D_{ON}^{on}, D_{ON}^{on}+D_{ON}^{off}, and D_{ALL} (*i.e.*, D_{ON}^{on}+ D_{ON}^{off} + D_{OFF}^{off}). For the D_{ON}^{on} detector, if a UE is flagged as positive by D_{ON}^{on} within any of the detection windows, it is deemed malware-infected. For the D_{ON}^{on} + D_{ON}^{off} detector, if a UE is flagged as positive by either D_{ON}^{on} or D_{ON}^{off} within any of the detection windows, the UE is treated to be malware-infected. For the D_{ALL} detector, if a UE is flagged as positive by either D_{ON}^{on} or D_{OFF}^{off} within any of the detection windows, it is deemed to be malware-infected.

Figure 5 reveals that the overall performance achieved by D_{ALL} reaches a false positive rate of 0.9 ± 1.8% and a detection rate of 95.7 ± 3.2% for CICAndMal2017-mixed and a false positive rate of 0.9 ± 0.7% and a detection rate of 94.7 ± 1.7% for ZooBazaar-mixed. For both datasets OMAD5G has achieved the targeted average false positive rate, which is below 1%. The detection performance differences seen in Figure 5 shows the different roles played by the



(1) CICAndMal2017-mixed



(2) ZooBazaar2024-mixed

Figure 5: Integrated detection performances of OMAD5G

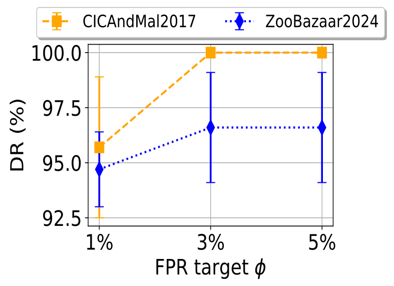


Figure 6: Effects of targeted false positive rate ϕ on detection performances

three individual detectors. For D_{ON}^{on}, although its false positive rate is low, its detection rate is only ∼31% for CICAndMal2017-mixed and ∼78% for ZooBazaar2024-mixed. D_{ON}^{off} is able to catch almost all malware-infected UEs for both datasets, but its false positive rate can be high: about 78% for CICAndMal2017-mixed and 25% for ZooBazaar2024-mixed. D_{OFF}^{off} is able to further improve the detection performances: for both datasets, it can detect ∼95% of the malware-infected UEs while having a false positive rate below 1%.

The tiered detection scheme adopted by OMAD5G avoids mirroring all UEs' mobile traffic to the heavyweight off-path malware detection module. From the experiments, we observe that the average fraction of UEs whose traffic is mirrored to the off-path detection module per detection window is 39.7% and 10.9% for the CICAndMal2017-mixed and ZooBazaar2024-mixed datasets, respectively. As parameter η is set to be 40% in our experiments, the results imply that its impact on tri-threshold learning is more significant with CICAndMal2017-mixed than that with ZooBazaar2024-mixed.

Effect of targeted false positive rate ϕ. We vary the targeted FPR ϕ among 1%, 3%, and 5% while keeping $\eta = 40\%$ and $\beta = 0.7$. Figure 6 shows how the detection rate changes with parameter ϕ. We notice that for both datasets, when ϕ increases from 1% to 3%, the detection rate also increases. This is unsurprising because a higher ϕ means that a higher false positive rate can be tolerated to achieve higher detection rates. Indeed, when $\phi = 1\%$, the observed detection rate is 95.7% for CICAndMal2017-mixed and 94.7% for ZooBazaar2024-mixed while the false positive rates are 0.9% for both datasets. When ϕ increases to 3%, the observed detection rate becomes 100% and 96.6% for CICAndMal2017-mixed and ZooBazaar2024-mixed, respectively, and the false positive rates are 2.7% and 2.9% for the two datasets, respectively. However, when we further increase ϕ from 3% to 5%, the detection rate does not change significantly any more. This is because even with increased ϕ, the observed false positive rates still stay the same for both datasets.

Effect of mirroring limit parameter η. The mirroring limit parameter η gives the maximum fraction of UEs whose traffic could be forwarded to the off-path detection module. We vary η from 10% to 100% while keeping $\phi = 1\%$ and $\beta = 0.7$. For the CICAndMal2017-mixed dataset, when η is less than 40%, the tri-threshold learning step fails to find proper thresholds that can achieve the targeted FPR of 1% based on the validation dataset. When η increases to 40%, a detection rate of 95.7% can be achieved on the testing dataset. When we further increase η to 50% or higher, OMAD5G can achieve a detection rate of 100%. For the ZooBazaar2024-mixed dataset, when $\eta = 10\%$, no proper thresholds can be found to achieve the targeted FPR of 1% based on the validation dataset. When we increase η to 20% or higher, the detection rate becomes stable at about 95%.

These results agree well with the experimental results shown in Figure 5. The domain name-based features have weaker predictive power in the CICAndMal2017-mixed dataset than them in the ZooBazaar2024-mixed dataset, as evidenced by the detection rate of D_{ON}^{on} for ZooBazaar2024-mixed, which is more than double of that for CICAndMal2017-mixed. Hence, to achieve high detection accuracy, a larger portion of traffic has to be mirrored to the off-path detection module for CICAndMal2017-mixed than for ZooBazaar2024-mixed. When there is sufficient traffic mirrored to the off-path detection module, further increasing parameter η affects little its detection performance for both datasets. The peak detection performance with sufficient mirrored mobile traffic is affected by the discriminatory power of the FC/PC/TS features when they are fed to the transformer neural network model. For CICAndMal2017-mixed, the peak detection rate of OMAD5G can reach almost 100% while for ZooBazaar2024-mixed, its peak detection rate stays at ∼95%.

Effect of overlapping ratio β. We vary β from 0.1 to 0.9 to study its effects. We set the targeted FPR to be below 1%. We also let the mirroring limit parameter η be 100% (i.e., no limit in traffic mirroring) to constrain its own impact on tri-threshold learning. Figure 8 shows how the detection rate is affected by parameter β. We notice that for both datasets, the optimal value for β is 0.7. If β is too low the training data can be more noisy because a detection window even with few malicious network activities are still labelled as positive. By contrast, if β is too high, only those detection windows containing intensive malicious network activities are labelled as positive; therefore, the detection model has weaker generalization capabilities to detect those malware-infected UEs with fewer malicious network activities.

8.4 Resource usage on P4 programmable switch

Due to limited TCAM and SRAM memory on the P4 programmable switch, the P4 compiler produces an error if the switch does not have enough resources to support all the PDU sessions. With the malware detection service enabled for all UEs, the UPF can have up to 124K PDU sessions on the Intel Tofino-based P4 programmable switch. Table 5 shows the amount of resources consumed by the UPF with and without OMAD5G. By comparing the first two rows, we observe that DNS packet processing for feature extraction significantly increases the resource usages by both Packet Header

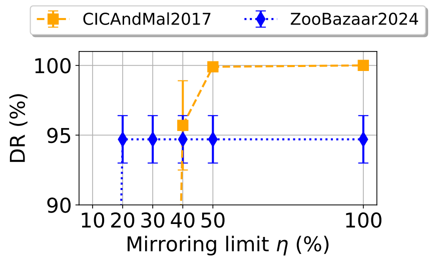


Figure 7: Effects of mirroring limit parameter η on detection performances

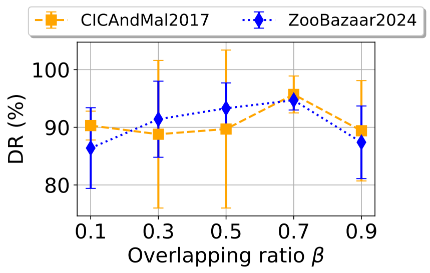


Figure 8: Effects of overlapping ratio β on detection performances

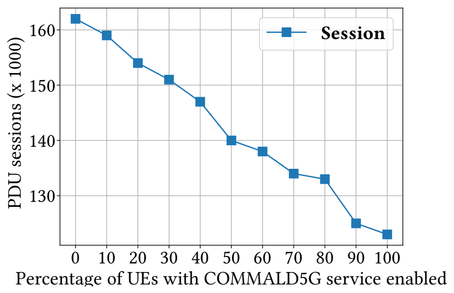


Figure 9: Number of PDU sessions with the fraction of UEs enabling OMAD5G

Table 5: Resource usage of OMAD5G. The first row shows resource usage with only UPF deployed with neither domain reputation score tracking nor DNS packet parsing. The second row shows resource usage with DNS packet parsing and hashing but no reputation score tracking. The last row is the case when the full OMAD5G service is deployed.

Service On?	No. of Domains	No. of Sessions	SRAM	PHV	Parser States
No	0	0	72.0%	37.0%	52.7%
Yes	0	0	75.5%	70.9%	73.8%
Yes	2×8192	124K	89.7%	70.6%	73.8%

Vectors (PHVs) and parser states. PHVs are temporary registers used to store fields extracted from packet headers. This observation is unsurprising as the monitor run by the switch ASIC needs to parse DNS packets and store extracted domain names.

The third row presents the resource usages when the full OMAD5G service is enabled for all 124K PDU sessions. In this case, malicious domain tracking is turned on and there are two malware classification models, each including reputation scores for 8192 simplified domain names. In comparison against the second row, we observe that these functionalities increase the SRAM usage by 18.8%. The extra SRAM usage results from the storage of domain reputation scores, mapping from PDU session IDs to indices of reputation scores, and the storage of session-level reputation scores.

We perform new experiments to study how the maximum number of PDU sessions supported on the UPF is affected by the fraction of UEs with the service enabled. The results are depicted in Figure 9. With no UE opting to enable the service, the maximum number of PDU sessions supported is 162K, which is 6% fewer than the 173K PDU sessions supported by the vanilla UPF. It is noted that even if no UE enables the malware detection service, resources are still allocated for DNS packet parsing and model storage. With an increasing fraction of UEs enabling malware detection, the maximum number of PDU sessions supported decreases almost linearly.

8.5 Execution performances

As the on-path malware detection module is performed by the switch ASIC of the P4 programmable switch at line speed, its execution performance is limited only by the hardware. We use the CICAndMal2017-mixed dataset to evaluate the execution performance of the off-path malware detection module. We assume that there are 5,000 UEs, each with one-hour mixed PCAP traces. We use Cisco's TRex tool [28] to replay these traces to the UPF in real time. We use two computer servers connected via two 100 Gigabit Ethernet cables, both of which are also connected to a Netberg Aurora 710 P4 programmable switch. One of the x86 server runs TRex and serves as the traffic generator for the UPF. The other server, which includes a 64-core Intel Xeon Gold 6338 CPU, 512GB RAM and three NVIDIA RTX A6000 GPUs, executes both the traffic collector and the off-path detector. Only one of the three GPUs, each having 48GB Video Random Access Memory (VRAM), is used.

We measure the serving times of both the traffic collector and the off-path malware detector. For each of the 5,000 UEs, we send request messages to the traffic collector and the off-path detector *sequentially* to prevent any waiting time from being included in our measurements. Our results show that the serving time of the traffic collector is 1.72 ± 0.53 milliseconds, data preprocessing takes 36.71 ± 17.26 milliseconds, and model inference by the off-path detector takes 6.65 ± 3.01 milliseconds with a peak VRAM usage of 7.3GB in the single GPU used in our experiments.

9 Conclusion

This work studies how to incorporate online malware detection as an integral service in 5G networks. OMAD5G combines fast DNS-based on-path malware detection and highly accurate off-path malware detection using transformer neural networks. We have implemented a prototype of OMAD5G based on an existing 5G network testbed. Our results show that OMAD5G presents a holistic solution to online malware detection in 5G networks that can tackle the accuracy, scalability, and integrality challenges effectively.

In the future we will address the following limitations of this study. Firstly, OMAD5G relies upon only domain name features for on-path malware detection. We will investigate other complementary features that can be extracted efficiently from mobile network traffic by P4 programmable switches to improve on-path malware detection accuracy and robustness. Secondly, our evaluation experiments used only two datasets in this study. We plan to collect more realistic datasets with a larger number of human users operating mobile apps on real devices. Lastly, as only Android mobile apps are considered in this study, we will extend this study by developing both on-path and off-path malware detectors for iOS mobile devices.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback and the shepherd for helping us in improving the final version of this paper. This work is partially supported by the U.S. National Science Foundation under award CNS-1943079.

References

[1] 2023. NetBeacon. https://github.com/IDP-code/NetBeacon. (2023).

[2] 2025. https://snort.org/. (2025).

[3] 2025. https://zeek.org/. (2025).

[4] 3GPP. 2018. *5G; Procedures for the 5G System.* Technical Specification (TS) 23.502. 3rd Generation Partnership Project (3GPP). https://www.etsi.org/deliver/etsi_ts/123500_123599/123502/15.03.00_60/ts_123502v150300p.pdf Version 15.3.0.

[5] 3GPP. 2021. *Security architecture and procedures for 5G system.* Technical Specification (TS) 33.501. 3rd Generation Partnership Project (3GPP). https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3169 Version 16.4.0.

[6] 3GPP. 2022. *Interface between the Control Plane and the User Plane Nodes.* Technical Specification (TS) 29.244. 3rd Generation Partnership Project (3GPP). https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3111 Version 17.3.0.

[7] abuse.ch. 2024. MalwareBazaar. https://bazaar.abuse.ch/. (2024).

[8] Mohammad Abuthawabeh and Khaled W Mahmoud. 2020. Enhanced Android malware detection and family classification, using conversation-level network traffic features. *The International Arab Journal of Information Technology* 17, 4A (2020), 607–614.

[9] Mujtahid Akon, Tianchang Yang, Yilu Dong, and Syed Rafiul Hussain. 2023. Formal Analysis of Access Control Mechanism of 5G Core Network. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security.* 666–680.

[10] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2016. Androzoo: Collecting millions of Android apps for the research community. In *Proceedings of the 13th international conference on mining software repositories.* 468–471.

[11] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. 2010. Building a dynamic reputation system for DNS. In *Proceedings of the 19th USENIX Security Symposium.*

[12] Manos Antonakakis, Roberto Perdisci, Wenke Lee, Nikolaos Vasiloglou II, and David Dagon. 2011. Detecting malware domains at the upper DNS hierarchy. In *Proceedings of the 20th USENIX Security Symposium.*

[13] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. 2012. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. In *Proceedings of the 21st USENIX Security Symposium.* 491–506.

[14] Anshul Arora and Sateesh K Peddoju. 2017. Minimizing network traffic features for Android mobile malware detection. In *Proceedings of the 18th International Conference on Distributed Computing and Networking.* 1–10.

[15] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. 2014. Drebin: Effective and explainable detection of Android malware in your pocket.. In *Network and Distributed System Security Symposium*, Vol. 14. 23–26.

[16] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. *ACM Sigplan Notices* 49, 6 (2014), 259–269.

[17] Ofek Bader, Adi Lichy, Chen Hajaj, Ran Dubin, and Amit Dvir. 2022. MalDIST: From encrypted traffic classification to malware traffic detection and classification. In *Proceedings of the 19th Annual Consumer Communications and Networking Conference.* IEEE, 527–533.

[18] David Basin, Jannik Dreier, Lucca Hirschi, Saša Radomirovic, Ralf Sasse, and Vincent Stettler. 2018. A formal analysis of 5G authentication. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security.* 1383–1396.

[19] Ramzi Bassil, Ali Chehab, Imad Elhajj, and Ayman Kayssi. 2012. Signaling oriented denial of service on LTE networks. In *Proceedings of the 10th ACM International Symposium on Mobility Management and Wireless Access.* 153–158.

[20] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. 2011. Exposure: Finding malicious domains using passive DNS analysis.. In *Proceedings of Network and Distributed System Security Symposium.* 1–17.

[21] Evangelos Bitsikas, Syed Khandker, Ahmad Salous, Aanjhan Ranganathan, Roger Piqueras Jover, and Christina Pöpper. 2023. UE Security Reloaded: Developing a 5G Standalone User-Side Security Testing Framework. In *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks.* 121–132.

[22] Giampaolo Bovenzi, Francesco Cerasuolo, Antonio Montieri, Alfredo Nascita, Valerio Persico, and Antonio Pescapé. 2022. A comparison of machine and deep learning models for detection and classification of Android malware traffic. In *Proceedings of IEEE Symposium on Computers and Communications.* IEEE, 1–6.

[23] Saurabh Chakradeo, Bradley Reaves, Patrick Traynor, and William Enck. 2013. Mast: Triage for market-scale mobile malware analysis. In *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks.* 13–24.

[24] Hsin-Fu Chang, Michael I-C Wang, Chi-Hsiang Hung, and Charles H-P Wen. 2022. Enabling malware detection with machine learning on programmable switch. In *Proceedings of IEEE/IFIP Network Operations and Management Symposium.* IEEE, 1–5.

[25] Rong Chen, Yangyang Li, and Weiwei Fang. 2019. Android malware identification based on traffic analysis. In *Proceedings of the 5th International Conference on Artificial Intelligence and Security.* Springer, 293–303.

[26] Zhenxiang Chen, Hongbo Han, Qiben Yan, Bo Yang, Lizhi Peng, Lei Zhang, and Jin Li. 2015. A first look at Android malware traffic in first few minutes. In *Proceedings of IEEE Trustcom/BigDataSE/ISPA*, Vol. 1. IEEE, 206–213.

[27] Chhaya Choudhary, Raaghavi Sivaguru, Mayana Pereira, Bin Yu, Anderson C Nascimento, and Martine De Cock. 2019. Algorithmically generated domain detection and malware family classification. In *International Symposium on Security in Computing and Communications.* Springer, 640–655.

[28] Cisco. 2024. TRex Low-Cost, High-Speed Stateful Traffic Generator. https://github.com/cisco-system-traffic-generator/trex-core. (2024).

[29] C Cremers and M Dehnel-Wild. 2019. Component-based formal analysis of 5G-AKA: channel assumptions and session confusion. In *Proceedings of Network and Distributed System Security Symposium.* Internet Society.

[30] Thiago A Navarro do Amaral, Raphael V Rosa, David F Cruz Moura, and Christian E Rothenberg. 2021. An in-kernel solution based on xdp for 5g upf: Design, prototype and performance evaluation. In *2021 17th International Conference on Network and Service Management (CNSM).* IEEE, 146–152.

[31] Christo El-Morr, Manar Jammal, Hossam Ali-Hassan, and W El-Hallak. 2022. Machine Learning for Practical Decision Making. *International Series in Operations Research and Management Science* (2022).

[32] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. 2014. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)* 32, 2 (2014), 1–29.

[33] William Enck, Patrick Traynor, Patrick McDaniel, and Thomas La Porta. 2005. Exploiting open functionality in SMS-capable cellular networks. In *Proceedings of the 12th ACM conference on Computer and communications security.* 393–404.

[34] Somayyeh Fallah and Amir Jalaly Bidgoly. 2022. Android malware detection using network traffic based on sequential deep learning models. *Software: Practice and Experience* 52, 9 (2022), 1987–2004.

[35] Kaiming Fang and Guanhua Yan. 2020. Paging storm attacks against 4G/LTE networks from regional Android botnets: rationale, practicality, and implications. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks.* 295–305.

[36] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. 2011. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices.* 3–14.

[37] Jiayin Feng, Limin Shen, Zhen Chen, Yuying Wang, and Hui Li. 2020. A two-layer deep learning method for Android malware detection using network traffic. *IEEE Access* 8 (2020), 125786–125796.

[38] Yu Feng, Saswat Anand, Isil Dillig, and Alex Aiken. 2014. Apposcopy: Semantics-based detection of Android malware through static analysis. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering.* 576–587.

[39] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM computing surveys (CSUR)* 46, 4 (2014), 1–37.

[40] Carrie Gates and Carol Taylor. 2006. Challenging the anomaly detection paradigm: A provocative discussion. In *Proceedings of The Workshop on New Security Paradigms.* 21–29.

[41] Mahshid Gohari, Sattar Hashemi, and Lida Abdi. 2021. Android malware detection and classification based on network traffic using deep learning. In *Proceedings of the 7th International Conference on Web Research.* IEEE, 71–77.

[42] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[43] Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. 2012. Riskranker: scalable and accurate zero-day Android malware detection. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services.* 281–294.

[44] GSMA. 2023. The 5G Era: How 5G is Changing the World. (September 2023). https://www.gsma.com/futurenetworks/networks-blog-series/the-5g-era-how-5g-is-changing-the-world/

[45] Josh Howarth. 2023. Internet Traffic from Mobile Devices (Sept 2023). (2023). https://explodingtopics.com/blog/mobile-internet-traffic#percentage-of-mobile-traffic
[46] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and J Doug Tygar. 2011. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*. 43–58.
[47] Syed Rafiul Hussain, Mitziu Echeverria, Imtiaz Karim, Omar Chowdhury, and Elisa Bertino. 2019. 5GReasoner: A property-directed security and privacy analysis framework for 5G cellular network protocol. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 669–684.
[48] Syed Ibrahim Imtiaz, Saif ur Rehman, Abdul Rehman Javed, Zunera Jalil, Xuan Liu, and Waleed S Alnumay. 2021. DeepAMD: Detection and identification of Android malware using high-efficient Deep Artificial Neural Network. *Future Generation computer systems* 115 (2021), 844–856.
[49] Roger Piqueras Jover and Vuk Marojevic. 2019. Security and protocol exploit analysis of the 5G specifications. *IEEE Access* 7 (2019), 24956–24963.
[50] Alexander Kaplan and Shir Landau Feibish. 2022. Practical handling of DNS in the data plane. In *Proceedings of the Symposium on SDN Research*. 59–66.
[51] Ângelo Cardoso Lapolli, Jonatas Adilson Marques, and Luciano Paschoal Gaspary. 2019. Offloading real-time DDoS attack detection to programmable data planes. In *Proceedings of IFIP/IEEE Symposium on Integrated Network and Service Management*. IEEE, 19–27.
[52] Arash Habibi Lashkari, Andi Fitriah A Kadir, Laya Taheri, and Ali A Ghorbani. 2018. Toward developing a systematic approach to generate benchmark android malware datasets and classification. In *Proceedings of the International Carnahan Conference on Security Technology*. IEEE, 1–7.
[53] Jehyun Lee and Heejo Lee. 2014. GMAD: Graph-based Malware Activity Detection by DNS traffic analysis. *Computer Communications* 49 (2014), 33–47.
[54] Patrick PC Lee, Tian Bu, and Thomas Woo. 2007. On the detection of signaling DoS attacks on 3G wireless networks. In *Proceedings of the 26th IEEE International Conference on Computer Communications*. IEEE, 1289–1297.
[55] Zaoxing Liu, Hun Namkung, Georgios Nikolaidis, Jeongkeun Lee, Changhoon Kim, Xin Jin, Vladimir Braverman, Minlan Yu, and Vyas Sekar. 2021. Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric DDoS attacks with programmable switches. In *Proceedings of the 30th USENIX Security Symposium*. 3829–3846.
[56] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).
[57] Robert MacDavid, Carmelo Cascone, Pingping Lin, Badhrinath Padmanabhan, Ajay Thakur, Larry Peterson, Jennifer Rexford, and Oguz Sunay. 2021. A P4-based 5G user plane function. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research*. 162–168.
[58] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083* (2017).
[59] Jyoti Malik and Rishabh Kaushal. 2016. CREDROID: Android malware detection by network traffic analysis. In *Proceedings of the 1st ACM Workshop on Privacy-Aware Mobile Computing*. 28–36.
[60] Khulood Al Messabi, Monther Aldwairi, Ayesha Al Yousif, Anoud Thoban, and Fatna Belqasmi. 2018. Malware detection using DNS records and domain name features. In *Proceedings of the 2nd International Conference on Future Networks and Distributed Systems*. 1–7.
[61] Jose Andre Morales, Areej Al-Bataineh, Shouhuai Xu, and Ravi Sandhu. 2010. Analyzing and exploiting network behaviors of malware. In *Proceedings of International ICST Conference on Security and Privacy in Communication Networks*. Springer, 20–34.
[62] Mozilla. 2020. PUBLIC SUFFIX LIST. (September 2020). https://publicsuffix.org/
[63] ONF. 2021. SD-FABRIC: Open Source Full-Stack Programmable Leaf-Spine Network Fabric. https://opennetworking.org/wp-content/uploads/2021/06/SD-Fabric-White-Paper-FINAL.pdf. (2021).
[64] Sourav Panda, KK Ramakrishnan, and Laxmi N Bhuyan. 2022. Synergy: A smartnic accelerated 5G dataplane and monitor for mobility prediction. In *2022 IEEE 30th International Conference on Network Protocols (ICNP)*. IEEE, 1–12.
[65] Attia Qamar, Ahmad Karim, and Victor Chang. 2019. Mobile malware attacks: Review, taxonomy & future directions. *Future Generation Computer Systems* 97 (2019), 887–909.
[66] Vaibhav Rastogi, Yan Chen, and William Enck. 2013. AppsPlayground: automatic security analysis of smartphone applications. In *Proceedings of the Third ACM conference on Data and Application Security and Privacy*. 209–220.
[67] Luca Schumann, Trinh Viet Doan, Tanya Shreedhar, Ricky Mok, and Vaibhav Bajpai. 2022. Impact of evolving protocols and COVID-19 on Internet traffic shares. *arXiv preprint arXiv:2201.00142* (2022).
[68] Clayton Scott. 2007. Performance measures for Neyman-Pearson classification. *IEEE Transactions on Information Theory* 53, 8 (2007), 2852–2863.
[69] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. 2012. "Andromaly": a behavioral malware detection framework for Android devices. *Journal of Intelligent Information Systems* 38, 1 (2012), 161–190.
[70] Deepak Sharma. 2016. Android malware detection using decision trees and network traffic. *International Journal of Computer Science and Information Technologies* 7, 4 (2016), 1970–1974.
[71] Robin Sommer and Vern Paxson. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of IEEE Symposium on Security and Privacy*. IEEE, 305–316.
[72] Kimberly Tam, Salahuddin J Khan, Aristide Fattori, and Lorenzo Cavallaro. 2015. Copperdroid: Automatic reconstruction of Android malware behaviors.. In *Proceedings of the Network and Distributed System Security Symposium*. 1–15.
[73] Xin Tong, Yang Feng, and Anqi Zhao. 2016. A survey on Neyman-Pearson classification and suggestions for future research. *Wiley Interdisciplinary Reviews: Computational Statistics* 8, 2 (2016), 64–81.
[74] Patrick Traynor, Chaitrali Amrutkar, Vikhyath Rao, Trent Jaeger, Patrick McDaniel, and Thomas La Porta. 2011. From mobile phones to responsible devices. *Security and Communication Networks* 4, 6 (2011), 719–726.
[75] Guan-Hua Tu, Chi-Yu Li, Chunyi Peng, Yuanjie Li, and Songwu Lu. 2016. New security threats caused by IMS-based SMS service in 4G LTE networks. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1118–1130.
[76] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Proceedings of Advances in Neural Information Processing Systems* 30 (2017).
[77] Zhixin Wen, Harsh Sanjay Pacherkar, and Guanhua Yan. 2022. VET5G: A Virtual End-to-End Testbed for 5G Network Security Experimentation. In *Proceedings of the 15th Workshop on Cyber Security Experimentation and Test*. 19–29.
[78] Zhixin Wen and Guanhua Yan. 2024. HiP4-UPF: Towards High-Performance Comprehensive 5G User Plane Function on P4 Programmable Switches. In *Proceedings of the 2024 USENIX Annual Technical Conference (ATC'24)*.
[79] Guorui Xie, Qing Li, Yutao Dong, Guanglin Duan, Yong Jiang, and Jingpu Duan. 2022. Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 1938–1947.
[80] Jinzhu Yan, Haotian Xu, Zhuotao Liu, Qi Li, Ke Xu, Mingwei Xu, and Jianping Wu. 2024. Brain-on-Switch: Towards Advanced Intelligent Network Data Plane via NN-Driven Traffic Analysis at Line-Speed. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 419–440.
[81] Bee Wah Yap and Chiaw Hock Sim. 2011. Comparisons of various types of normality tests. *Journal of Statistical Computation and Simulation* 81, 12 (2011), 2141–2155.
[82] Menghao Zhang, Guanyu Li, Shicheng Wang, Chang Liu, Ang Chen, Hongxin Hu, Guofei Gu, Qianqian Li, Mingwei Xu, and Jianping Wu. 2020. Poseidon: Mitigating volumetric DDoS attacks with programmable switches. In *Proceedings of the 27th Network and Distributed System Security Symposium (NDSS 2020)*.
[83] Yuan Zhang, Min Yang, Bingquan Xu, Zhemin Yang, Guofei Gu, Peng Ning, X Sean Wang, and Binyu Zang. 2013. Vetting undesirable behaviors in Android apps with permission use analysis. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications Security*. 611–622.
[84] Guangmeng Zhou, Zhuotao Liu, Chuanpu Fu, Qi Li, and Ke Xu. 2023. An efficient design of intelligent network data plane. In *Proceedings of the 32nd USENIX Security Symposium*.

Appendix A: Notations of detection models

Table 6 summarizes the notations of different malware detection models used by OMAD5G.

Table 6: Explanations of detection models used by OMAD5G

Notation	Explanation
D_{ON}	On-path detection model trained to predict malware infection likelihood using objective function in Eq. (2)
D_{ON}^{on}	On-path detection model using LFP-threshold Θ_{ON}^{on}
D_{ON}^{off}	On-path detection model using LFN-threshold Θ_{ON}^{off}
D_{OFF}	Off-path detection model (a transformer neural network) for predicting malware infection likelihood
D_{OFF}^{off}	Off-path detection model using threshold Θ_{OFF}^{off}

Appendix B: Android system domains

Table 7 summarizes the list of domain names used by the Android system by default.

Table 7: List of domains used by the Android system by default

Domain Name	Usage
google.com	Google
googleapis.com	Google
gstatic.com	Google
googleusercontent.com	Google
android.com	Google
gvt1.com	Google
gvt2.com	Google
googleadservices.com	Google Ads
google-analytics.com	Google Ads
doubleclick.net	Google Ads
app-measurement.com	App service provided by Google
crashlytics.com	App service provided by Google
googletagmanager.com	App service provided by Google
ntp.org	Time sync
youtube.com	Youtube
ytimg.com	Youtube
tenor.com	Android keyboard GIFs

Appendix C: Effects of feature types and model hyperparameters on off-path malware detection

A new set of experiments with the same settings as in Section 8.2 are performed to evaluate the effects of different feature types and model hyperparameters on off-path malware detection. We examine the contributions of each individual type of features (i.e., FC, PC, or TS) to the detection accuracy measured in F-1 scores and also study whether combining these different types of features can help improve the detection accuracy. When PC features are used, our transformer neural network architecture uses at least 80 bytes to ensure that they can at least cover an IP header (at least 20 bytes), a UDP header (at least 8 bytes), and some DNS request and response contents. Hence, we let N_{PC} be a multiple of 80 bytes in our experiments.

The results with the CICAndMal2017 and ZooBazaar2024 datasets are summarized in Table 8 and 9, respectively. We make the following key observations. For the CICAndMal20217 dataset, using only FC features leads to a good F1-score (97.03%) and adding TS features improves the F1-score slightly by 1.3%; however, additional PC features do not improve the detection accuracy. For the ZooBazaar2024 dataset, using only FC features leads to an F1 score of 85.24% and adding TS features has little effect on the detection accuracy; by contrast, increasing N_{PC} for the PC features helps improve the F1 score. These observations suggest that using the combination of FC, PC, and TS features for off-path malware detection leads to a more robust model which can perform well in diverse situations.

Table 8: Comparison of feature choices in the off-path detection model for the CIC-AndMal2017 dataset. Our final model choice is highlighted with *.

Features	F1 (%)
FC	97.03 ± 0.51
PC[N_{PC} = 160]	95.36 ± 1.60
FC+PC[N_{PC} = 160]	96.91 ± 0.66
FC+TS	98.31 ± 0.46
FC+PC[N_{PC}=80]+TS	98.19 ± 1.31
FC+PC[N_{PC}=160]+TS*	98.19 ± 0.64
FC+PC[N_{PC}=240]+TS	97.82 ± 1.10
FC+PC[N_{PC}=320]+TS	98.07 ± 0.45

Table 9: Comparison of feature choices in the off-path detection model for the ZooBazaar2024 dataset. Our final model choice is highlighted with *.

Features	F1 (%)
FC	85.24 ± 2.68
PC[N_{PC} = 160]	87.84 ± 2.13
FC+PC[N_{PC} = 160]	88.58 ± 1.28
FC+TS	85.38 ± 2.55
FC+PC[N_{PC} = 80]+TS	87.53 ± 2.40
FC+PC[N_{PC} = 160]+TS*	88.28 ± 2.31
FC+PC[N_{PC} = 240]+TS	88.69 ± 2.45
FC+PC[N_{PC} = 320]+TS	89.91 ± 2.29

Appendix D: Implementation details

User Safety Function (USF). Apart from the code that is needed to serve as a 5G network function like registering itself in the NRF, the USF exposes the following APIs to the external NFs:

- POST /nusf_training/v1/pcap: Add a pcap file to its traffic trace database;
- POST /nusf_training/v1/train: Manually trigger model training for a given model;
- GET /nusf_training/v1/train: Query the progress of model training for a given model if training is still running;
- GET /nusf_training/v1/result: Query the results of model training such as test accuracy for a given model;
- POST /nusf_training/v1/config: Update training configurations, such as the learning rate;
- GET /nusf_training/v1/config: Get training configurations;
- POST /nusf_deploy/v1/deploy: Initiate the deployment of a given model.

UPF. The OMAD5G controller is built upon the HiP4-UPF's controller [78] with an additional Rust module handling OMAD5G-related tasks.

During UPF startup we populate the domain suffix count table based on the domain suffix list curated by Mozilla [62].

In a 5G network when a UE goes idle for a certain period of time, the core network can trigger a connection suspension procedure to suspend its PDU sessions in order to save resources. This procedure removes the states of the PDU sessions in not only the base station based also the UPF. The connection is resumed when some downlink packet arrives or the UE wakes up. To ensure a seamless malware detection experience, when a PDU session is suspended, OMAD5G saves its stored features from switch dataplane onto the switch DRAM. This frees up the valuable SRAM in the ASIC for other active PDU sessions. These features are restored back to the ASIC if the connection is resumed.

In a 5G UPF, the Packet Detection Rule ID (PDR_ID), which is identified in the UPF based on packet headers, is used to decide how the packet should be forwarded. This ID is carried over from the ingress to the egress stage so it can be used by the egress stage to look up the *detection_id* (18-bit) that indexes the table storing the aggregate reputation score for each PDU session. This ID is also used in the ingress stage as key to the mirroring table.

SMF. For SMF, apart from adding OMAD5G rule installation code to the PDU Session Establishment/Modification procedures, we add a new API for model deployment:

- POST /nsmf_c5g/v1/deploy: Deploy a classification model to the UPF controlled by this SMF.

N4 interface. The N4 interface in a 5G network connects SMF and UPF by exchanging Packet Forwarding Control Protocol (PFCP) messages [6]. A PFCP message consists of one or more Information Elements (IEs). We modify these messages to support OMAD5G features.

Detection enablement is implemented as part of usage reporting. In a 5G network usage reporting is triggered when data quota are reached or by certain events such as the first packet arrival. As we view malware alerts as another type of events, we use existing PFCP messages to enable malware detection and notify malware infection.

Specifically, we add a special REPort OMAD5G (REP5G) flag as the 25-th bit of the ReportingTriggers IE. This flag, along with the MeasurementMethod IE set to EVENT and the MeasurementPeriod IE set to the detection period, enables the UPF to create the OMAD5G context for the corresponding PDU session.

Once malware infection is detected, an alert is sent by the UPF to the SMF in a UsageReport IE within a N4 Session Report Request message. The UsageReport IE contains the information needed to identify the triggering PDU session and a timestamp.

For model deployment we opt to use new PFCP message types. We added two new types of N4 messages: PFCP Model Deployment Request and its response. In the request message we define three IEs: ModelAction, ModelWeightsBlob and ReputationScores. The ModelAction IE is used to enable/disable a model and set the malware detection thresholds. The ModelBinaryBlob IE is used to transmit the weights used by the off-path detection module. The ReputationScores IE contains a list of simplified domain names and their reputation scores used by the on-path detection module. The ModelAction IE contains the identifiers of the malware classification models and flags instructing to enable or to disable the models. If all the IEs cannot fit into a single UDP packet due to a large size of the models, we send multiple Deployment Requests to deploy a single malware classification model.

Appendix E: Domain name statistics

The on-path malware detection module uses simplified domain names as features to detect malicious mobile traffic. Table 2 shows that for both CICAndMal2017 and ZooBazaar2024 datasets, the average number of distinct simplified domain names requested by a malware app is about three times of that requested by a benign one. One concern is that if a malicious app visits mostly previously unseen DNS names, the on-path malware detector becomes incapable of detecting its traffic as malicious. To study how often a UE can visit a previously unseen domain name during the testing phase, we create a mixed dataset based on randomly partitioned five folds of individual PCAP traces. As tri-threshold learning is unnecessary here, we do not use a validation dataset. Using four folds for training and one for testing, we generate 1000 four-hour-long mixed traffic traces for training and 500 for testing for each mixing ratio α varied from 0 to 4.

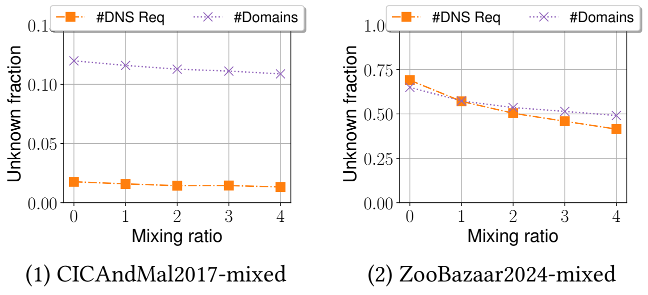


Figure 10: Unknown simplified domain names during the testing phase. #DNS Req: fraction of DNS request packets carrying unknown simplified domain names; #Domains: fraction of distinct unknown simplified domain names.

Figure 10 shows the fraction of DNS request packets carrying UNKNOWN simplified domain names and the fraction of distinct UNKNOWN simplified domain names during the testing phases for both datasets. The general trend is that as more benign pcaps are included in the mixed dataset both the fraction of DNS request packets carrying UNKNOWN simplified domain names and the fraction of distinct UNKNOWN simplified domain names decrease. This observation agrees well with our intuition that it is more likely for malware to visit short-lived domain names with techniques such as Domain Generation Algorithms (DGAs) [13]. Between the two datasets, the fraction of distinct UNKNOWN simplified domain names is much higher with the ZooBazaar-mixed dataset than that with the CICAndMal2017-mixed dataset. For example, when there is no benign PCAP trace included (i.e., $\alpha = 0$), about 71% of the simplified domain names requested by a UE never appear in the training dataset for the ZooBazaar-mixed dataset, while only 12% of them never appear in the training dataset for the CICAndMal2017-mixed dataset. This observation suggests that the DNS domain names requested in the ZooBazaar dataset are much more diverse than those in the CICAndMal2017 dataset. Indeed, the total numbers of distinct simplified domain names requested by all benign and malicious apps are only 1747 and 3363, respectively, in the CICAndMal2017 dataset, while they are 2892 and 26609, respectively, in the ZooBazaar dataset.

Appendix F: Further discussions

Both on-path and off-path malware detection models should be constantly adapted to address natural concept drift [39] or adversarial machine learning attacks [46]. Their robustness can be enhanced by retraining with new data or adversarial training [42, 58]. Adaption of these models in dynamic or adversarial environments is out of the scope of this work.

Although currently OMAD5G considers 5G UPFs deployed on P4 programmable switches, its DNS-based on-path malware detection module can be easily implemented in other UPF deployments such as those based on smartNICs [64] and eBPF/XDP [30]. Reputation score quantization may be unnecessary as these platforms can deal with floating number operations directly.