EdgeShield: Enabling Collaborative DDoS Mitigation at the Edge

Xiaoyu Xia, *Member, IEEE*, Feifei Chen, *Senior Member, IEEE*, Qiang He, *Senior Member, IEEE*, Ruikun Luo, Bowen Liu, Caslon Chua, Rajkumar Buyya, *Fellow, IEEE*, and Yun Yang, *Senior Member, IEEE*

Abstract—Edge computing (EC) enables low-latency services by pushing computing resources to the network edge. Due to the geographic distribution and limited capacities of edge servers, EC systems face the challenge of edge distributed denial-of-service (DDoS) attacks. Existing systems designed to fight cloud DDoS attacks cannot mitigate edge DDoS attacks effectively due to new attack characteristics. In addition, those systems are typically activated upon detected attacks, which is not always realistic in EC systems. DDoS mitigation needs to be cohesively integrated with workload migration at the edge to ensure timely responses to edge DDoS attacks. In this paper, we present EdgeShield, a novel DDoS mitigation system that leverages edge servers' computing resources collectively to defend against edge DDoS attacks without the need for attack detection. Aiming to maximize system throughput over time without causing significant service delays, EdgeShield monitors service delays and migrates workloads across an EC system with adaptive mitigation strategies. The experimental results show that EdgeShield outperforms state-of-the-art solutions by 27.17%-78.55% in terms of system throughput and 58.12%-66.00% in terms of service delays.

Index Terms—Edge computing, DDoS attack, mitigation.

1 Introduction

Edge Computing (EC) offers groundbreaking opportunities for workload management, which has been intensively studied in cloud computing [31], [35]. Figure 1 illustrates an EC system comprised of four edge servers that may receive both benign and attack workloads from nearby users in a metropolitan area, e.g., smart web cameras, autonomous vehicles, etc. These devices may offload workloads onto nearby edge servers for processing with low service latency [26]. Furthermore, this helps alleviate the traffic burden on the back-haul network [29]. However, the geographic distribution of edge servers in an EC system raises many new security issues. In particular, a distributed denial-of-service (DDoS) attack in EC may control a number of devices to generate attack workloads against edge servers [18]. For example, with Mirai - a malware used to attack web servers belonging to PayPal, Netflix, Twitter, and Spotify in 2016, an adversary can control over 400,000 devices as bots to coordinate a DDoS attack [9]. The rapid growth of mobile and Web-of-Things (WoT) devices in recent years significantly increases the number of potential bots that can be leveraged to attack the edge servers in an EC system.

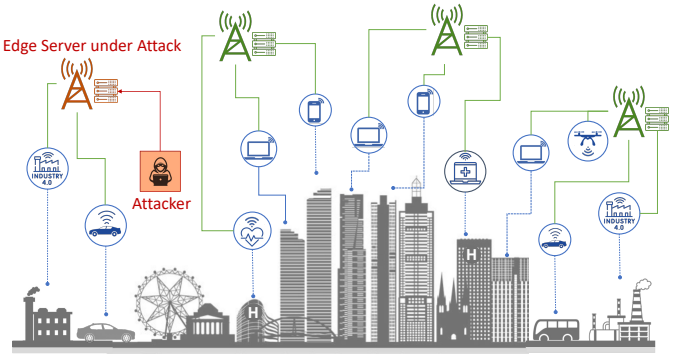


Fig. 1. An example EC system. The system consists of multiple edge servers and edge devices such as smart vehicles, smartphones, laptops, etc.

DDoS attacks against cloud servers have been studied intensively [1], [3], [37], but not those against edge servers at the network edge [18]. Many previous studies have focused on detecting DDoS attacks by differentiating attack workloads from benign ones [41], [49]. However, it typically requires long-term workload monitoring, which is unsuitable in EC systems that often host dynamic and short-lived services [14] (§2). With virtually infinite resources available in the cloud, recent DDoS defense mechanisms

- *X. Xia is with the School of Computing Technologies, RMIT University, Australia. E-mail: xiaoyu.xia@rmit.edu.au.*
- *F. Chen is with the School of Information Technology, Deakin University, Australia. E-mail: feifei.chen@deakin.edu.au.*
- *Q. He is with the National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, China, and also with the Department of Computing Technologies, Swinburne University of Technology, Melbourne, VIC 3122, Australia. E-mail: hqiang@hust.edu.au.*
- *R. Luo is with the National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, China. E-mail: rkluo@hust.edu.cn.*
- *B. Liu is with the State Key Laboratory for Novel Software Technology, Nanjing University, China, and iSING Laboratory, The Hong Kong University of Science and Technology, Hong Kong. E-mail: liubw@smail.nju.edu.cn.*
- *C. Chua and Y. Yang are with the Department of Computing Technologies, Swinburne University of Technology, Australia. E-mail: {cchua, yyang}@swin.edu.au.*
- *R. Buyya is with the School of Computing and Information Systems, The University of Melbourne, Australia. E-mail: rbuyya@unimelb.edu.au.*

have employed workload migration to mitigate DDoS attacks, making it a resource competition between the attacker and the defender [13], [18]. The main idea is to migrate both benign and attack workloads across autoscaled virtual machine instances so that they can all be processed timely with minimum resources without the need for attack detection [3]. However, compared with powerful cloud servers, edge servers suffer from constrained resources [8]. They cannot be autoscaled easily and flexibly like cloud servers to handle migrated workloads. As a result, they fall victim to DDoS attacks easily, which can cause significant economic and societal losses because edge servers usually power critical applications like autonomous driving [24]. Other cloud DDoS defense solutions based on traffic scrubbing [13] and routing orchestration [40] are not practical for EC systems either because their responses also take seconds to minutes. In fact, the slow responses of state-of-the-art DDoS defense solutions have inspired the new pulse-wave DDoS attacks [1].

Fortunately, with peer-offloading [8], edge servers in the same area can communicate with each other and transmit data efficiently [42]. They can mitigate DDoS attacks collaboratively [18]. In fact, even without attacks, jobs often need to be migrated across edge servers for processing to utilize their resources collectively [46]. This allows edge DDoS mitigation to be integrated with job migration cohesively without having to turn on edge servers' intrusive "defense mode" upon detected attacks. This paper presents EdgeShield, a novel DDoS mitigation system that leverages edge servers' collective resources to mitigate edge DDoS attacks collaboratively without the need for attack detection. It is designed to maximize system throughput over time while ensuring low service latency under edge DDoS attacks. The main contributions include:

- We design EdgeShield to fight edge DDoS attacks specifically, considering the unique attack characteristics and system characteristics.
- We propose a *cumulative delay monitor* (CDM) to ensure low service delay in EdgeShield.
- We design a *job completion estimator* (JCE) to estimate job completion times, so that the *adaptive time scaler* (ATS) can dynamically adjust the time slot length for formulating mitigation strategies.
- We design an *attack mitigation engine* (AME) for formulating mitigation strategies with the support of CDM and ATS.
- We experimentally evaluate the performance of EdgeShield and the results demonstrate that EdgeShield outperforms all benchmark systems significantly.

2 Background and Related Work

In cloud computing, DDoS attacks have been studied comprehensively from various perspectives [36], e.g., detecting attacks based on attack characteristics [3], mitigating attacks through job migration mechanisms [13], etc. Edge DDoS attacks are structurally different from cloud DDoS attacks. In a cloud DDoS attack, attack workloads usually come from attackers worldwide [13]. While cloud DDoS attacks usually target a single server in the cloud, edge DDoS attacks can target multiple edge servers in the same area at the network edge, compromising their ability to support each other and crippling the defense mechanisms deployed at the core of the internet. In addition, low service latency is a key feature promised and pursued by EC [15]. To accommodate edge users' dynamic demands with low end-to-end latency, services are often deployed in docker containers on edge servers for quick starts with minimal overheads [39]. Thus, edge services are usually short-lived and it is challenging to monitor their workloads constantly for DDoS attack detection. Even if constant workload monitoring is possible, the workload analysis for attack detection usually takes a long time, up to 15 seconds according to recent studies [49]. Similarly in industry, existing edge DDoS protections are also based on attack detection, such as Cisco Secure DDoS Edge Protectionprovided by Cisco, and an edge DDoS solution called Autonomous Edgeprovided by CloudflareMost defense mechanisms, including Cisco Secure DDoS Edge Protection and Autonomous Edge, also take seconds to minutes to respond to an attack, which is the major vulnerability exploited by the new pulse-wave DDoS attack [1]. Given the need to maintain low service latency for users, edge servers cannot afford to wait for a long time to receive a detection result before taking necessary mitigation actions.

Edge servers are especially vulnerable to DDoS attacks due to the constrained resources which can be rapidly exhausted by an edge DDoS attack [8]. A straightforward solution is to offload the workloads to the cloud. However, many users would experience high service latency, which conflicts with their low latency requirement for edge services. Fortunately, edge servers can share their resources through the high-speed communication links connecting them [8] to perform various tasks. The workloads on edge servers under an edge DDoS attack can be offloaded to nearby edge servers for processing. In this way, the edge servers can mitigate such DDoS attacks collectively. This is the key idea of existing edge DDoS mitigation approaches [18], [20], [48]. In [20], SecEG is proposed to defend against the edge DDoS attack by implementing a queueing theory model for resource allocation on edge servers. In addition, SecEG applies the anomaly detection way to roughly identify benign and attack workloads, then isolates the attack workloads in an isolated container. In [18], the authors propose EDMGame, a game-theoretic approach designed to mitigate edge DDoS attacks. When an attack is detected, it kicks in and transfers the workloads of the affected edge servers to other edge servers for processing. However, as mentioned earlier, detecting DDoS attacks in EC systems is a challenging task, which makes it difficult to implement SecEG and EDMGame in practice. Zhou et al. [48] introduce a prediction-free online approach (PFO) by using the resources in both cloud and edge to mitigate edge DDoS attacks to balance the workloads mitigated to edge servers and remote cloud. However, the above-mentioned approaches aim to maximize the current system throughput without considering system performance over time, suffering from poor performance against continuous DDoS attacks (§5).

Integrating edge DDoS mitigation with runtime job migration can 1) ensure low service latency by fast responses; and 2) alleviate the need for DDoS attack detection. This mo-

tivates the design of EdgeShield. Please note that edge DDoS mitigation is fundamentally different from conventional load balancing. Load balancing aims to achieve vastly different goals, e.g., fair workload distribution, equal remaining capacities, etc., with "evenness" as a key objective [47]. To mitigate edge DDoS attacks, EdgeShield tries to fully utilize the capacities of edge servers close to the edge servers under attack.

3 Edge DDoS Mitigation Problem

In EC, devices can submit attack or benign workloads to nearby edge servers with latency constraints. When an edge DDoS attack occurs, adversaries generate and submit a large volume of jobs to target edge servers as attack workloads. As discussed in §2, the accurate and timely detection of edge DDoS attacks is difficult. Fortunately, the resources on edge servers can be shared to collaboratively process workloads. Thus, collaboration among edge servers is critical to mitigating edge DDoS attacks. The notations and symbols used in this paper are summarized in 1.

TABLE 1
Summary of Notations

Notation	Description
$c_{i,r}^t$	resource r needed by job j_i^t in time slot t
$c_{m,r}^{t,a}$	amount of available resource r on edge server s_m in time slot t
r	resource r
R	set of resources
$\mathcal{L}_{CM}$	cumulative service delay over time
$\mathcal{L}_{CM}^t$	cumulative service delay by a time slot t
$l_i^{t,E}$	latency constraint for j_i^t in time slot t
l_i^t	actual service latency of j_i^t
M	number of edge servers
S	set of edge servers
s_m	mth edge server
t	time lot $t \in T$
T	set of time
J	set of jobs over time
J^t	set of jobs in time slot t
j_i^t	ith job in time slot t
σ^t	mitigation strategy for time slot t
$\sigma_{i,m}^t$	binary value indicating whether job j_i^t is allocated to s_m in time slot t

System Throughput. Given M edge servers in an EC system, denoted by $S = \{s_1, ..., s_m, ..., s_M\}$, and jobs arrived in time slot $t \in T$, denoted by J^t, the objective of the edge DDoS mitigation problem is to maximize the system throughput for migrating the stress of an edge DDoS attack. This is quantified by the number of benign jobs processed during the mitigation process. In time slot t, the system throughput can be formulated by $\sum_{j_i^t \in J_B^t} \sum_{s_m \in S} \sigma_{i,m}^t$, where J_B^t is the set of newly arrived benign jobs in time slot t and $\sigma_{i,m}^t \in \{0, 1\}$ is the mitigation decision about whether job j_i^t is allocated to edge server s_m in time slot t or not. If job j_i^t is allocated to edge server s_m in time slot t, there is $\sigma_{i,m}^t = 1$; otherwise, there is $\sigma_{i,m}^t = 0$. The mitigation decisions for all the jobs constitute the mitigation strategy in time slot t, denoted by σ^t.

Capacity Constraint. Since each edge server only has a limited amount of resources, e.g., CPU, GPU, memory, bandwidth, etc., denoted by R, each type of resource $r \in R$ required by all the jobs newly assigned to an edge server in any time slot must not exceed its current available resources, i.e., $\sum_{j_i^t \in J^t} c_{i,r}^t \sigma_{i,m}^t \leq c_{m,r}^{t,a}, \forall s_m \in S, r \in R, t \in T$, where $c_{i,r}^t$ is the amount of resource $r \in R$ needed by job j_i^t and $c_{m,r}^{t,a}$ is the available amount of r on edge server s_m in time slot t. Please note that edge servers are heterogeneous in their resources, as well as their runtime workloads.

Latency Constraint. Each job comes with its own latency constraint $l_i^{t,E}$. Thus, j_i^t's service latency should be no longer than $l_i^{t,E}$. To fulfill this latency constraint, $l_i^t \leq l_i^{t,E}, \forall j_i^t \in J^t, t \in T$ should be fulfilled, where l_i^t is the actual service latency for job j_i^t according to the mitigation strategy σ^t. Specifically, the service latency is calculated by the data size of job j_i^t and the allocated bandwidth, such as inter-edge bandwidth via edge mitigation and cloud-edge bandwidth via cloud mitigation.

Problem Formulation. The edge DDoS mitigation problem can be formulated with the aim to maximize the system throughput over time T, while fulfilling capacity constraint and latency constraint.

Threat Model. In this study, attackers can coordinate a DoS attack by sending attack jobs to a subset of edge servers. However, they cannot gain direct control over edge servers to sabotage their collaborative defense.

4 EdgeShield System

4.1 System Overview

Figure 2 overviews an EdgeShield system. Before EdgeShield runs the attack mitigation engine (AME) to formulate mitigation strategies to solve the edge DDoS mitigation problem effectively, it first collects the current system status, such as the service latency and processing time of the jobs finished in previous time slots. After that, the cumulative delay monitor (CDM) calculates the cumulative delay to check the latency constraint and sends the result to the AME. In this way, EdgeShield keeps the low service delay in long-term via CDM. In the meantime, the job completion estimator (JCE) estimates the completion time of each current job based on the processing time of completed jobs. Based on JCE's estimation, the adaptive time scaler (ATS) decides the time slot length to determine the timing for running AME. With JCE and ATS, EdgeShield can utilize available resources effectively with low system overhead. When the AME starts to formulate the mitigation strategy, it collects information about new jobs in the time slot, such as resource demands, latency constraints, and arrival time. Based on the value of the cumulative service latency provided by the cumulative delay monitor, the AME formulates the mitigation strategy to decide where each job goes, and to which edge server. If the system resources are insufficient to mitigate all the jobs, the AME mitigates partial jobs to the remote cloud. We will now introduce CDM (§4.2), JCE (§4.3), ATS (§4.4), and AME (§4.5) in detail.

4.2 Cumulative Delay Monitor

To maximize system throughput, a straightforward solution is to process as many jobs as possible without considering

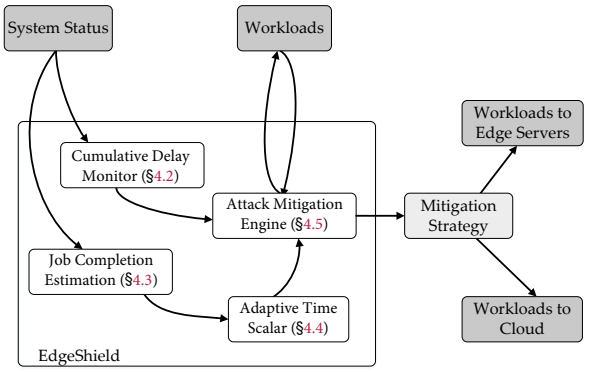


Fig. 2. EdgeShield system overview. EdgeShield consists of four main components, including the cumulative delay monitor (CDM), job completion estimator (JCE), adaptive time scaler (ATS) and attack mitigation engine (AME).

their latency constraints. However, this solution may result in a lot of late job completions and violates the corresponding service level objectives (SLOs). To calculate and manage the service delay, we introduce the cumulative job delay, i.e., the total delay in processing all previous jobs. According to latency constraint in §3, the cumulative service latency, denoted by $\mathcal{L}_{CM}$, can be calculated by $\mathcal{L}_{CM} = \left\lfloor \sum_{t\in T}\sum_{j_i^t\in J^t} \sigma_{i,m}^t \cdot (l_i^t - l_i^{t,E}) \right\rfloor_+$. Then, the average service delay $\mathcal{L}_{avg}$ in an EC system is $\mathcal{L}_{avg} = \frac{\mathcal{L}_{CM}}{\sum_{t\in T}\sum_{j_i^t\in J^t}\sigma_{i,m}^t}$. Since EdgeShield aims to eliminate the service latency to ensure the low latency services in the long-term, this service delay should eventually converge to 0 over time T, i.e., $\mathcal{L}_{avg} = 0, t \to |T|$. However, this requires that the complete information about system dynamics must be available in advance, i.e., all the jobs over T and edge servers' remaining capacities in every time slot. Unfortunately, it is not realistic in a real-world EC system where jobs arrive dynamically.

To tackle this challenge, EdgeShield employs a cumulative delay monitor (CDM) to monitor the current cumulative delay for AME (§4.5). The cumulative delay by a time slot t can be obtained by $\mathcal{L}_{CM}^t = \left\lfloor \mathcal{L}_{CM}^{t-1} + \sum_{j_i^{t-1}\in J^{t-1}} \sigma_{i,m}^t \cdot (l_i^{t-1} - l_i^{t-1,E}) \right\rfloor_+$. The cumulative delay by time slot t is calculated based on the cumulative delay by time slot $t-1$ and the total job delay in time slot $t-1$. This indicates that when the cumulative delay increases, the value of $\mathcal{L}_{CM}^t$ increases. When $\mathcal{L}_{CM}^t$ is more than 0, CDM sends this value to the AME for triggering strategy adjustment (§4.5).

4.3 Job Completion Estimator

As discussed in §4.1, ATS in EdgeShield relies on the estimated completion times of the current jobs. Accurate estimation will contribute to proper utilization of edge servers' capacities against edge DDoS attacks. Many approaches have been proposed in recent years to make workload-related predictions based on machine learning models [19]. However, these approaches are likely to suffer from poor performance in estimating job completion times for edge servers. The main reasons are twofold: 1) the user demands are highly dynamic and often vary drastically across edge servers [32]; and 2) edge servers' resources are heterogeneous and limited [46]. It is challenging to train a machine learning model universally suitable for every edge server in different areas. More importantly, high efficiency and low overheads in formulating mitigation strategies must be ensured in the EC environment, while the training process of a machine learning model is relatively slow [49]. To tackle those challenges, EdgeShield employs a lightweight algorithm JCE to estimate job completion times rapidly.

Algorithm 1: Job Completion Estimator (JCE)

Input: the processing time records of the latest completed jobs on every edge server and the start time $\lambda_{star,i}$ of each current job j_i^t
Output: the estimated completion time $E[\lambda_{comp,i}]$ of each current job j_i^t

1 **for** *current job* j_i^t **do**
2 read processing time records of similar completed jobs on edge server s_m processing j_i^t
3 find processing time $\bar{time}_{proc,m}$ of best matching workload on s_m
4 calculate the ratio α_m of the longest job processing time over $\bar{time}_{proc,m}$ according to the records
5 read job j_i^t's elapsed processing time $time_{proc,i}^{curr}$
6 **if** $time_{proc,i}^{curr} \le \bar{time}_{proc,m}$ **then**
7 $E[\lambda_{comp,i}] = \lambda_{star,i} + \bar{time}_{proc,m}$
8 **else**
9 $E[\lambda_{comp,i}] = \lambda_{star,i} + rand(time_{proc,i}^{curr}, \alpha_m \cdot time_{proc,i}^{curr})$

JCE processes the current jobs iteratively. For a current job j_i^t, JCE obtains the processing time of similar completed jobs on the edge server processing j_i^t (Line 2). Then, JCE employs a signature-based prediction method [23] to find the job most similar to j_i^t in resource demands, and obtains its processing time $\bar{time}_{proc}$ (Line 3). It is possible that j_i^t's elapsed processing time has already exceeded $\bar{time}_{proc}$. In such cases, JCE also calculates the ratio of the longest job processing time over $\bar{time}_{proc}$, denoted by α_m, for estimating j_i^t's completion time (Line 4). According to j_i^t's elapsed processing time (Line 5), denoted by $time_{proc,i}^{curr}$, JCE checks whether $time_{proc,i}^{curr}$ is longer than the best matching processing time $\bar{time}_{proc}$ (Line 6). If there is $time_{proc,i}^{curr} \le \bar{time}_{proc}$, JCE assigns $\bar{time}_{proc}$ as the estimated completion time of j_i^t (Line 7), assuming that job j_i^t's processing time is similar to the best matching job previously completed on the same edge server. If j_i^t's elapsed processing time is already longer than $\bar{time}_{proc}$, j_i^t is likely to take more time to complete. In this case, JCE randomly selects an estimated completion time from $1\times$ to $\alpha_m \times time_{proc,i}^{curr}$ for j_i^t (Line 9).

JCE Complexity. Let $J_{m,comp}^t$ denote the number of jobs completed on edge server s_m in time slot t. In this case, the computation complexities of Line 2 and Line 3 in time slot t' are $O(\sum_{t\in t'} J_{m,comp}^t)$. Since the computation complexities of Lines 4-9 are always constant, i.e., $O(1)$, the average complexity of JCE over time is $O(\frac{\sum_{t\in T} J_{m,comp}^t}{T}) \approx O(\frac{|J|}{MT})$.

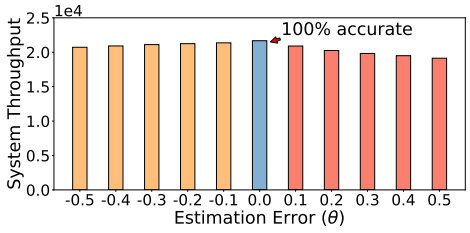
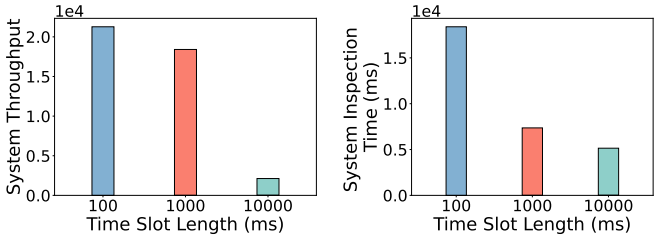


Fig. 3. System throughput $v.s.$ estimation error (θ) on 20 edge servers over 100 seconds. The experiments were executed under the default settings presented in §5.1.



(a) System throughput $v.s.$ $|t|$ (b) System inspection time $v.s.$ $|t|$

Fig. 4. EdgeShield with different fixed time slot lengths $|t|$.

Here, We conducted a set of experiments to investigate the impact of JCE's estimation on EdgeShield's performance. Aiming for high efficiency and low overheads, JCE may not always be able to estimate job completion times accurately. Inaccurate estimation may mislead ATS in determining the right timing for mitigation strategy formulation. We conducted a set of experiments to investigate the impact of JCE's inaccurate estimation on the performance of EdgeShield. In the experiments, we manually fed incorrect estimates of job completion times to ATS. The estimation error (θ) ranges from -0.5 to 0.5 in steps of 0.1. For example, if the actual completion time of a job is 200ms, an estimate with a -0.5 error is 100ms, and an estimate with a 0.5 error is 300ms. We ran the experiments under the default settings presented in §5.1. Figure 3 illustrates the results. Unsurprisingly, EdgeShield achieves the highest system throughput when the estimates of job completion times are always accurate ($\theta = 0$), with a 13.28% advantage over the worst case ($\theta = 0.5$). Figure 3 shows that the performance of EdgeShield decreases with the increase in estimation error ($|\theta|$). In addition, Table 2 also summarizes the estimation errors of JCE under five attack distributions including sinusoidal distribution (SINUS), exponential distribution (EXPO), poisson distribution (POISS), gamma distribution (GAMMA) and pulse-wave distribution (PW), during the experiments in §5.3. The results demonstrate that JCE's θ is ranged from 0.2 to 0.4 under various attacks. Please note that once a more accurate estimator is available, EdgeShield can directly integrate it to improve the performance.

TABLE 2
JCE's average estimation error (θ) under five attacks.

	SINUS	EXPO	POISS	GAMMA	PW
θ	0.25	0.35	0.34	0.18	0.39

4.4 Adaptive Time Scaler

Most existing studies of dynamic problems in cloud computing and EC assume an equal length for all the time slots and execute algorithms in each time slot. Usually, job arrivals follow a specific distribution in general, e.g., sinusoidal distribution [43], uniform distribution [5], etc. In this case, job arrivals can be estimated based on the corresponding distribution. It is reasonable to consider a fixed length for all the time slots if server capacities are known. However, an edge DDoS attack does not follow these distributions because it generates a large volume of attack jobs within a short period of time in a geographical area. When the attack occurs, a fixed time slot length can easily cause system resource waste because edge servers' idle resources cannot be utilized timely. Here, we analyze the disadvantages of fixed-length time slots in two cases and present the Adaptive Time Scaler as a solution.

Case 1: Short Fixed Time Slots. In this case, small time slots are considered, e.g., 100ms. With such small time slots, timely system inspection allows high system resource utilization and high system throughput when an edge DDoS attack occurs. For example, running over 100 seconds under the same implementation settings in §5.1, EdgeShield's system throughput with different fixed time slot lengths, i.e., 100ms, 1,000ms and 10,000ms in Figure 4(a). EdgeShield with 100ms time slots achieves the highest system throughput, i.e., processing 0.16× and 10.01× more jobs than EdgeShield with 1,000ms and 10,000ms time slots. As discussed in §4.1, EdgeShield needs to inspect the system status before formulating the mitigation strategy. Frequent inspections will consume more time overall. As shown in Figure 4(b), the total time for EdgeShield to inspect the system status with 100ms time slots is 18,393ms during an edge DDoS attack, higher than 7,361ms with 1,000ms time slots and 5,146ms with 10,000ms time slots. This indicates that frequent system status inspection consumes edge servers' bandwidth, slows down the mitigation process, and increases service latency.

Case 2: Long Fixed Time Slots. In this case, large time slots are considered, e.g., 10,000ms. Most jobs can be processed within one time slot. This often leads to system resources' under-utilization because a lot of resources remain idle before the next time slot. This is evidenced by the low system throughput presented in Figure 4(a) when the time slot length is 10,000ms.

As discussed above, the time slot length significantly impacts the performance of EdgeShield in mitigating an edge DDoS attack. Unfortunately, it is impossible to find an optimal time slot length without all the job and system information over time, even without the DDoS attack. EdgeShield employs an Adaptive Time Scaler (ATS) algorithm to adjust the time slot length. Its pseudo code is presented in Algorithm 2.

At the beginning, ATS checks the current time slot length denoted by I^t, and the end of the current time slot denoted by λ^t. ATS also inspects whether any jobs have been finished and how many jobs have been offloaded to the remote cloud. If no jobs have been finished or more than half of new jobs have been offloaded to the remote cloud, ATS

Algorithm 2: Adaptive Time Scaler (ATS)

Input: current time slot length I^t and I^t's end timestamp λ^t
Output: next time slot length I^{t+1}

1 **if** *no jobs have been finished or more than half of new jobs have been offloaded to the remote cloud in time slot* t **then**
2 $I^{t+1} = \frac{I^t}{2}$
3 **else**
4 obtain the median value $E[\lambda_{comp,medi}]$ from the estimated completion times of all the current jobs and the median ratio $\alpha_{medi} = \text{median}\{\alpha_m | \forall s_m \in S\}$ by JCE, where α_m is the ratio of the longest job processing time over the average job processing time on edge server s_m
5 **if** $E[\lambda_{comp,medi}] - \lambda^t \geq I^t \cdot \alpha_{medi}$ **then**
6 $I^{t+1} = I^t \cdot \alpha_{medi}$
7 **else**
8 **if** $E[\lambda_{comp,medi}] - \lambda^t \geq \frac{I^t}{\alpha_{medi}}$ **then**
9 $I^{t+1} = \frac{\max\{E[\lambda_{comp,medi}] - \lambda^t, I^t\}}{\alpha_{medi}}$
10 **else**
11 $I^{t+1} = \frac{I^t}{\alpha_{medi}}$

reduces the length of the next time slot I^{t+1} to $\frac{I^t}{2}$ (Line 2). This is because the current jobs' processing time has exceeded a time slot and they are likely to be finished soon. If there are finished jobs, ATS executes JCE to estimate the current jobs' completion times and computes the ratio α on each edge server. Then, ATS obtains the median value $E[\lambda_{comp,medi}]$ of all the estimated completion times and the median value α_{medi} of the ratios (Line 4). Next, ATS uses the median value of the estimated remaining time $E[\lambda_{comp,medi}] - \lambda^t$ to determine the length of the next time slot I^{t+1}. When this median value is higher than $\alpha_{medi} \times I^t$, it indicates that most of the current jobs are likely to be finished after running over a period of $\alpha_{medi} \times I^t$ time. As observed in Figure 4(a), a longer time slot will lower the system throughput. Thus, ATS sets the length of the next time slot to $\alpha_{medi} \times I^t$ in this case (Line 6). When $E[\lambda_{comp,medi}] - \lambda^t$ is in the range of $[\frac{1}{\alpha_{medi}}, \alpha_{medi}] \times I^t$, ATS sets $I^{t+1} = \frac{\max\{E[\lambda_{comp,medi}] - \lambda^t, I^t\}}{\alpha_{medi}}$ to leverage the advantages of a short time slot while avoiding excessively long system inspection time (Line 9). For the same reason, when $E[\lambda_{comp,medi}] - \lambda^t$ is lower than $\frac{I^t}{\alpha_{medi}}$, ATS sets $I^{t+1} = \frac{I^t}{\alpha_{medi}}$ (Line 11).

ATS Complexity. According to Algorithm 2, the average computation complexity of Line 1 is $(\frac{\sum_{t \in T} \sum_{s_m \in M} J^t_{m,comp}}{T}) = O(\frac{|J|}{T})$. The computation complexity of Line 4 is $O(\sum_{s_m \in S} 1 \cdot \frac{|J|}{TM})$, given the $O(\frac{|J|}{MT})$ complexity of JCE. In addition, the computation complexity of Lines 5-11 is $O(1)$. Thus, the computation complexity of ATS is $\max\{O(\frac{|J|}{T}), O(\sum_{s_m \in S} 1 \cdot \frac{|J|}{TM})\} = O(\frac{|J|}{T})$.

To evaluate the performance of ATS in the EdgeShield system, we implement an implementation of EdgeShield without ATS, named EdgeShield$_{-ATS}$, as a benchmark system in §5.2. According to Figure 4, 1,000ms is a proper time slot length for EdgeShield$_{-ATS}$. Thus, we set the time slot length to 1,000ms in the evaluation.

4.5 Attack Mitigation Engine

Edge servers are geographically distributed [7] and remote control from the cloud introduces significant delays in their responses to DDoS attacks [46]. Edge servers must coordinate to mitigate edge DDoS attacks collaboratively. A series of game-theoretic approaches have been designed to enable collaboration among edge servers [12], [18]. They require up to hundreds of communication rounds among edge servers to make a major decision. Even with a small EC system, this can lead to a high delay in edge DDoS mitigation and edge servers cannot act until the mitigation strategy is formulated. To tackle this challenge, EdgeShield employs a global algorithm named AME (Attack Mitigation Engine) to migrate all the jobs, benign or attack.

Algorithm 3: Attack Mitigation Engine (AME)

Input: system information, new jobs, and current jobs in time slot t
Output: the mitigation strategy σ^t

1 normalize all the jobs' resource requirements based on the remaining resources in the system
2 sort all the jobs in ascending order of required resources based on the Euclidean norm
3 obtain the cumulative delay $\mathcal{L}^t_{CM}$ by current time slot t from CDM
4 $\mathcal{L}'^t_{CM} = \frac{\mathcal{L}^t_{CM}}{2}$
5 **for** *each sorted job* j^t_i **do**
6 **if** *no available edge server within* j^t_i*'s latency constraint* $l^{t,E}_i$ **then**
7 offload j^t_i to the remote cloud
8 **else**
9 **if** $\mathcal{L}'^t_{CM} > 0$ **then**
10 offload j^t_i to the edge server s_m with the minimum service latency $l^t_{i,m}$
11 $\mathcal{L}'^t_{CM} = \mathcal{L}'^t_{CM} - (l^t_i - l^{t,E}_{i,m})$
12 **else**
13 offload j^t_i to the edge server s_m with the most available resources based on the Euclidean distance between normalized resource requirements of j^t_i and normalized available resources on s_m

After ATS outputs the time slot length for the next time slot, AME runs at the end of the next time slot and collects system information as its input. AME migrates jobs heuristically, aiming to mitigate an edge DDoS attack. To maximize system throughput, AME needs to sort the jobs by the amount of required resources in ascending order. However, a job normally requires multi-dimensional resources, e.g., CPU and memory. Thus, before the sorting process, all the resource requirements are normalized based on the remaining system resources (Line 1). For example, let us assume that a job requires 0.2GB memory while the total remaining memory in the system is 10GB. It is normalized to 0.02. After that, we can sort all the jobs based on the Euclidean norm over all the resource dimensions (Line 2). The job requiring the least resources is migrated for processing first. Then, AME receives the cumulative service latency $\mathcal{L}^t_{CM}$

from CDM and assigns the value of $\frac{\mathcal{L}_{CM}^t}{2}$ to $\mathcal{L}_{CM}^{'t}$ (Lines 3 - 4). Reducing the cumulative delay to 0 in one time slot may require migrating many jobs to the remote cloud. Thus, AME will try to reduce the cumulative delay by half. Next, AME migrates the sorted jobs iteratively (Lines 6 - 13) based on the value of $\mathcal{L}_{CM}^{'t}$. In each iteration, AME first checks whether there is an edge server with sufficient resources for processing job j_i^t with a latency constraint $l_i^{t,E}$ (Line 6). Since the latency between edge servers may vary over time, we use the latency obtained in the previous time slot here. If no edge servers have sufficient resources due to the resource limit or exhaustion, this job is migrated to the remote cloud for processing (Line 7). Otherwise, this job is migrated to a suitable edge server depending on the value of $\mathcal{L}_{CM}^{'t}$ (Lines 9 - 13). When $\mathcal{L}_{CM}^{'t} > 0$, the cumulative delay is overly large. In this case, AME migrates the job to the edge server with the minimum service latency $l_{i,m}^{t,E}$ (Lines 10). Then, AME deducts the value of $\mathcal{L}_{CM}^{'t}$ by the reduced delay compared to j_i^t's latency constraint $l_i^{t,E}$ (Line 11). When $\mathcal{L}_{CM}^{'t}$ is equal to or less than 0, AME migrates the job to the edge server with the most available resources (Line 13).

AME Complexity. Since AME needs to sort the jobs in Line 2, the computation complexity of Line 2 depends on the selected sorting algorithm. In the implementation, the merge sort is adopted in this paper, and its computation complexity is $O(\frac{|J|}{T} \cdot log\frac{|J|}{T})$. For the iteration process in Lines 5-13, the computation complexity is $O(\frac{|J|}{T} \cdot M)$. Thus, the computation complexity of AME is $O(\frac{|J|}{T} \cdot \max\{M, log\frac{|J|}{T}\})$.

Since the complexities of JCE and ATS are $O(\frac{|J|}{MT})$ and $O(\frac{|J|}{T})$, respectively, the complexity of EdgeShield is $\max\{O(\frac{|J|}{MT}), O(\frac{|J|}{T}), O(\frac{|J|}{T} \cdot \max\{M, log\frac{|J|}{T}\})\} = O(\frac{|J|}{T} \cdot \max\{M, log\frac{|J|}{T}\})$, same as the complexity of ATS.

Remark. Once the attack detection is available, attack detection can complement EdgeShield by reducing the workloads transferred across edge servers. For example, high-risk workloads can be discarded immediately to reduce the overall system resources needed to mitigate an edge DDoS attack and to decrease the average service latency.

4.6 System Deployment

4.6.1 Leader Election

A conventional way to deploy EdgeShield is to run everything on a remote cloud server, including the CDM, JCE, ATS and AME. However, the high end-to-end network latency between the edge and the cloud will inevitably slow down EdgeShield's responses to edge DDoS attacks. The remote cloud server may also become a performance bottleneck or a potential vulnerability. For example, targeted attacks may be launched against the cloud server running EdgeShield to delay or block its communication with edge servers.

As discussed in §3, in the defense against an edge DDoS attack against edge servers in an area, EdgeShield leverages the resources available on other edge servers in the same area to ensure that benign jobs can be processed under corresponding latency constraints. Taking this into account, EdgeShield can be deployed on a specific edge server in the area to coordinate the edge servers' fight against an edge DDoS attack. This approach has been widely adopted to solve various problems at the edge, including the state-of-the-art solution to edge DDoS mitigation [18].

However, the edge server selected as the leader may also create a performance bottleneck and is potentially subject to targeted attacks. To tackle it, in each time slot, EdgeShield runs a secret election process to identify leaders to formulate mitigation strategies. Inspired by Algorand [16], the election process employs the bilinear mapping function (BMF) [2] and verifiable random function (VRF) [27] to protect an elected leader against targeted attacks by hiding its identity until it announces the mitigation strategy. A brief introduction to BMF and VRF can be found in §A.

In each time slot t, every edge server will implement the mitigation strategy σ^t received from the leader and set a timeout based on the length of the current time slot determined by the leader's ATS. Then, they will broadcast their workload information, including the completion times of their jobs in the current time slot, the requirements for their ongoing and new jobs, and the processing time of their ongoing jobs. Upon receiving the workload information from other edge servers, they run the AME to formulate a new mitigation strategy σ^{t+1}. Next, they participate in the election process for time slot $t+1$, which goes through two main phases, i.e., leadership bidding and leadership verification.

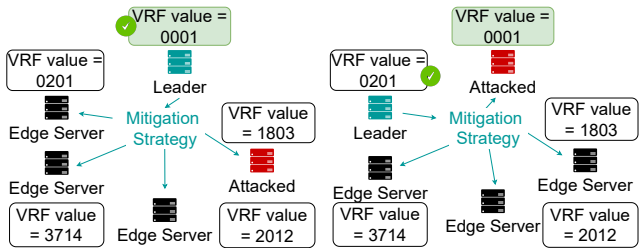
Leadership Bidding. To bid for leadership, every edge server s_m runs the VRF to create its own bid $bid_m =< tag_{key_{pri,m}}(seed), \ value_{key_{pri,m}}(seed), key_{pub,m}, \sigma_m^{t+1} >$, where $key_{pri,m}$ and $key_{pub,m}$ are private and public keys of m, and $seed$ is a random seed in VRF. When the timeout elapses, s_m sends its bid to other edge servers.

Leadership Verification. When an edge server s_n receives a bid, say bid_m, it verifies the correctness of bid_m with the BMF. If s_n receives multiple correct bids, it accepts the one with the lowest VRF value, say, bid_c, and implements the corresponding mitigation strategy σ_c^{t+1}.

Deployed on all the edge servers, EdgeShield eliminates the performance bottleneck created by a fixed leader, as well as the single-point failures. In each time slot, every edge server will formulate a mitigation strategy. The mitigation does not rely on any individual edge server. A potential drawback is the increasing communication overheads when the system scales up. Fortunately, it is difficult, if not impossible, to coordinate an edge DDoS attack against a large number of edge servers in an area because they are only accessible from within their serving areas [18].

4.6.2 Security Analysis

In the EC environment, edge servers are more vulnerable due to their limited resource. In practice, mitigation strategies may fail when the leader becomes unavailable or overwhelmed during targeted attacks. In each time slot, a leader is elected in EdgeShield and an edge server can infer whether it is elected as a leader by inspecting the received bids without a voting process. Since the VRF value is randomly generated based on a random $seed$, it is computationally difficult for adversaries to calculate other edge servers' VRF values. In this way, the attacker cannot identify the leader until its mitigation is released and the leadership



(a) Non-leader under attack (b) Leader under attack

Fig. 5. System under a targeted attack.

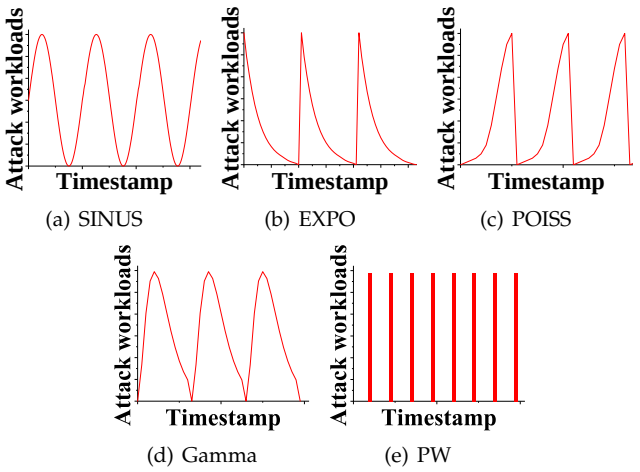


Fig. 6. DDoS attack patterns

can be protected from targeted attacks. Compared with Raft [28], the overhead of this leader election mechanism is significantly lower, thanks to its independence of a Raft-like voting process.

However, adversaries can still launch edge DDoS attacks targeting more edge servers, regardless of whether they function as a leader or not. In fact, EdgeShield can protect the entire system from targeted attacks in three general cases, as shown in Figure 5:

- **Case 1**: Non-leader edge servers are under attack. In this case, the mitigation strategy formulated by the leader will be implemented in the system for the edge servers in the system to mitigate the attack collectively.
- **Case 2**: The leader is under attack. The leader, the edge server with the lowest VRF value, may be slowed down or overwhelmed. In this case, another leader election process kicks off after a leader timeout, similar to Raft [28]. In this way, the mitigation strategy formulated by the new leader will be acknowledged and implemented in the system for the edge servers in the system to mitigate the attack collectively.
- **Case 3**: Both the leader and non-leader edge servers are under attack. In this case, the mitigation strategy formulated by the new leader will be acknowledged and implemented, same as **Case 2**.

According to the above analysis, EdgeShield possesses the capability to formulate mitigation strategies to defend against edge DDoS attacks no matter the leader edge server is specifically targeted or not. EdgeShield's effectiveness and efficiency in devising such strategies are key strengths that contribute to its resilience in protecting EC systems from edge DDoS attacks.

5 Evaluation

To evaluate EdgeShield, we implement a prototype and conduct intensive experiments to answer the following research questions:

- **Q1 Performance.** Is EdgeShield capable of mitigating different types of DDoS?
- **Q2 Scalability.** Does EdgeShield scale?
- **Q3 Robustness.** How does EdgeShield accommodate DDoS attacks of different intensities?
- **Q4 Overhead.** Is the overhead of EdgeShield acceptable?

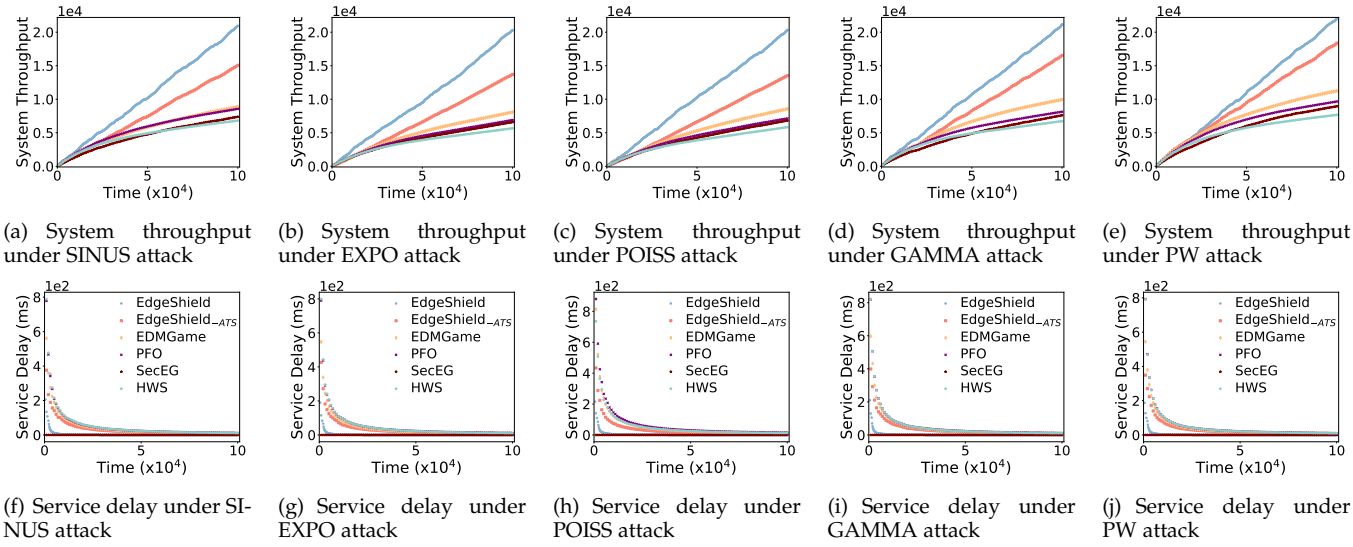
5.1 Environment Setup

Implementation. The prototype of EdgeShield is implemented on an EC system comprised of 30 edge servers running Ubuntu 16.04. Each edge server is equipped with 2, 4 or 8 vCPUs and 4, 8, 16 or 32 GB memory, the same as the edge servers provided by AWS Wavelength. Adjacent edge servers are linked based on their locations according to a given edge density, i.e., the number of links per edge server in the EC system. An r6g.12xlarge EC2 running Amazon Linux 2 with 48 vCPUs and 384 GB memory is deployed in Amazon's cloud as the cloud server. The inter-edge bandwidth is set as 1Gbps, same as the settings in [22], to ensure that the mitigation process is not throttled by the cloud server.

By default, EdgeShield is installed on 20 edge servers with an edge density of 2.0, and 6 of these edge servers are under attack. In the scalability and robustness tests (§5.4 and §5.5), the edge density, the number of edge servers and the edge DDoS attack intensity are varied to create various edge DDoS scenarios and EC system structures to evaluate the impacts of system scalability and system robustness. As discussed in §4.4, we set the time slot length to 1,000 milliseconds and run each experiment over 100 seconds. On average, each job, benign or attack, requires 0.20 CPUs and 0.66GB memory. We ran a stress test and found that on average, an edge server can process at most 20 jobs simultaneously. Accordingly, the number of new benign jobs arriving on an edge server in each time slot is randomly selected following the normal distribution $N(20, 1)$.

Job Generation. In the experiments, benign and attack jobs are generated based on resource usages/requirements obtained from the widely-used Alibaba Cluster Trace Program[1] [25]. We generate job information like job processing time by YCSB [10] for 200,000 jobs with arrival timestamps and edge server IDs. In the experiment, we randomly select edge servers from this synthesized dataset. Five typical types of attacks are launched, following the patterns shown in Figure 6, including the sinusoidal distribution (SINUS) [6], [38], exponential distribution (EXPO) [11], [33], poisson distribution (POISS) [45], gamma distribution (GAMMA) [34] and pulse-wave distribution (PW) [1]. As discussed in §2, the volume of an edge DDoS attack is much smaller than conventional cloud DDoS attacks. In

1. https://github.com/alibaba/clusterdata



(a) System throughput under SINUS attack
(b) System throughput under EXPO attack
(c) System throughput under POISS attack
(d) System throughput under GAMMA attack
(e) System throughput under PW attack
(f) Service delay under SINUS attack
(g) Service delay under EXPO attack
(h) Service delay under POISS attack
(i) Service delay under GAMMA attack
(j) Service delay under PW attack

Fig. 7. System performance under 5 different attacks at 6 random edge servers out of 20 over 100 seconds.

the experiments, an edge DDoS attack lasts 10 time slots, following the threat model in §3. In each time slot, each edge server under attack will receive a maximum of 60 jobs under a SINUS attack, 120 jobs under an EXPO attack, 80 jobs under a POISS attack, 60 jobs under a GAMMA attack and 70 jobs under a PW attack. Our stress test revealed that those attack jobs can easily exhaust an edge server.

The dataset synthesized in the experiments has been published[2] for the validation and reproduction of experimental results.

5.2 Benchmark Systems

To evaluate EdgeShield, five benchmark systems are implemented for comparison.

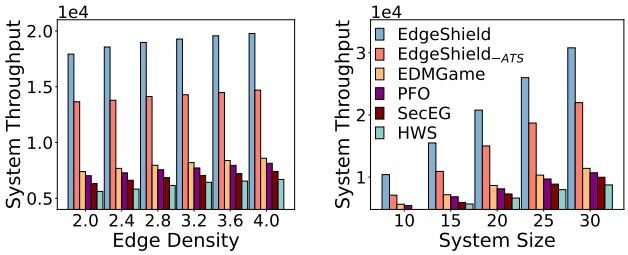
- **EdgeShield**$\_{ATS}$: This approach is a variant of EdgeShield without ATS.
- **EDMGame** [18]: In each time slot, EDMGame finds a Nash equilibrium as the mitigation strategy. The optimization objective is to minimize the overall mitigation cost, calculated based on the service latency and the penalty of unprocessed jobs. Unlike EdgeShield, EDMGame is activated when an edge DDoS attack is detected, which is impractical in real-world scenarios as discussed in §2. Thus, in the experiments, EDMGame runs in every time slot to formulate a mitigation strategy for both benign and attack jobs.
- **PFO** [48]: In each time slot, PFO first inspects the current system status and the information about newly arrived jobs. After that, it heuristically migrates those newly arrived jobs to edge servers with the minimum cost consisting of the mitigation cost and the processing cost as its final strategy in the current time slot. In our experiment, the cost is measured by the time taken for mitigation and processing. To facilitate a fair comparison, PFO employs the same leadership mechanism as EdgeShield (§4.6) for system deployment.
- **SecEG** [20]: SecEG implements a lightweight anomaly detection based on the Poisson distribution. In each time slot, SecEG first inspects the current edge server status and the information about newly arrived jobs on this edge server. Then, SecEG implements the detection and separates the jobs into two categories including whitelist and greylist. After that, SecEG allocates the available resources on an edge server equally to the jobs on the whitelist and greylist. SecEG migrates the jobs that cannot be executed on edge servers to the remote cloud.
- **HWS** [26]: In each time slot, HWS first finds a set of available mitigation strategies based on the Karush–Kuhn–Tucker (KKT) conditions [17], aiming to maximize the system throughput without considering the service delays. After that, HWS selects the mitigation strategy that utilizes the available resources on edge servers with the minimum service delay as its final strategy in the current time slot. Similar to PFO, HWS also employs the leadership mechanism presented in §4.6 for system deployment.

5.3 Overall Performance

The performance of EdgeShield is evaluated under five typical types of edge DDoS attacks, including SINUS-based, EXPO-based, POISS-based, GAMMA-based and PW-based edge DDoS attacks. The results answer ***Q1***.

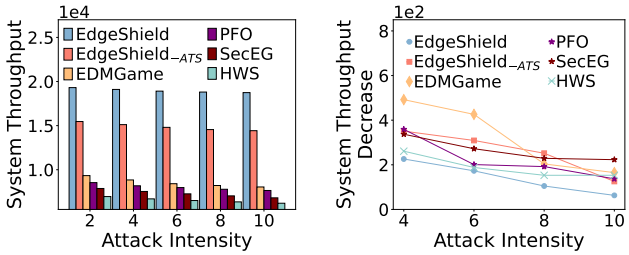
Under SINUS-based edge DDoS attacks, Figure 7(a) and Figure 7(f) show that EdgeShield achieves the highest system throughput and the lowest service delay. In terms of system throughput, EdgeShield has a 27.94% advantage over EdgeShield$_{ATS}$, 135.73% over EDMGame, 144.46% over PFO, 183.79% over SecEG and 207.43% over HWS. It can also be seen that the system throughput achieved by EdgeShield is continuously higher than that achieved by other approaches over time. Figure 7(f) demonstrates that the service delay in EdgeShield is much lower than those in other systems. In fact, EdgeShield is the only system that manages to converge the service delay to 0 under SINUS attacks. Figure 7(b) and Figure 7(g) show that EdgeShield can mitigate EXPO attacks, and outperform benchmark systems with large margins. Specifically, its system through-

2. https://anonymous.4open.science/r/dataset-7B08



(a) System throughput v.s. edge density (b) System throughput v.s. system size

Fig. 8. System scalability



(a) System throughput (b) Throughput decrease

Fig. 9. System robustness against attack intensity

put is 1.48×, 2.51×, 2.94×, 3.08× and 3.58× achieved by EdgeShield$_{-ATS}$, EDMGame, PFO, SecEG and HWS, respectively. Again, EdgeShield is the only system that fulfills the long-term latency constraint $\mathcal{L}_{avg} = 0,\ t \to |T|$ under EXPO attacks.

With similar performance and advantages over benchmark systems, Figure 7(c), Figure 7(h), Figure 7(d), Figure 7(i), Figure 7(e) and Figure 7(j) show that EdgeShield can mitigate POISS, GAMMA and PW attacks effectively. Under POISS, GAMMA and PW attacks, EdgeShield achieves an average of 1.51×, 1.28× and 1.20× system throughput achieved by EdgeShield$_{-ATS}$, 2.38×, 2.12× and 1.96× achieved by EDMGame, 2.85×, 2.60× and 2.28× achieved by PFO, 2.98×, 2.78× and 2.46× achieved by SecEG, and 3.49×, 3.14× and 2.87×of achieved by HWS, respectively. This demonstrates the superior performance of EdgeShield in mitigating edge DDoS attacks. In Figure 7, POISS attacks are the most difficult to be mitigated. Under POISS attacks, all five systems achieve the lowest throughput. However, the advantages of EdgeShield under POISS attacks are the most significant among all five DDoS attacks.

In summary, EdgeShield excels in mitigating edge DDoS attacks. It can mitigate all five typical types of edge DDoS attacks effectively with similar performance. In the remainder of this section, we illustrate and analyze its performance under SINUS attacks for demonstration purposes.

5.4 System Scalability

In practice, scalability is a crucial concern - EdgeShield must be able to scale with system size, which is the first scalability dimension. As discussed and illustrated in §1, an EC system is comprised of multiple connected edge servers. Their connectivity, often measured by the average number of edges per edge server, plays a crucial role in the performance of the system, particularly when edge servers need to collaborate. With a high edge density, edge servers can seek help from more edge servers under the latency constraint while fighting an edge DDoS attack. Similar to many studies [18], [21], [30], [44], we vary the edge density (measured by the number of links per edge server) from 2.0 to 4.0 in steps of 0.4 and the system size (measured by the number of edge servers in the system) from 10 to 30 in step 5 to investigate the scalability of EdgeShield. The results in Figure 8 answer *Q2*.

Figure 8(a) shows the results when varying the edge density from 2.0 to 4.0. EdgeShield achieves the highest system throughput again, with a 34.16% advantage over EdgeShield$_{-ATS}$, 136.62% over EDMGame, 149.76% over PFO, 175.17% over SecEG and 206.18% over HWS. Figure 8(b) reveals that EdgeShield again outperforms all other systems with significant margins. On average, its system throughput is 40.37% higher than EdgeShield$_{-ATS}$, 139.58% higher than EDMGame, 153.80% higher than PFO, 181.40% higher than SecEG and 208.69% higher than HWS. With the increase in system size from 10 to 30, the system throughput achieved by EdgeShield, EdgeShield$_{-ATS}$, EDMGame, PFO, SecEG, and HWS also increases by 196.15%, 189.65%, 102.40%, 97.09%, 112.41%, and 95.15%, respectively. We can see that EdgeShield scales with system size and can properly leverage the increase in system resources to mitigate edge DDoS attacks.

As shown in Figure 8, EdgeShield outperforms other comparison systems in scalability, evidenced by its much higher system throughput, as well as faster system throughput increases, compared with the other five approaches.

5.5 System Robustness

An edge DDoS attack tailored specifically for EdgeShield can be performed against multiple nearby edge servers simultaneously, making it difficult for edge servers to mitigate the attack collaboratively. Here, we vary the attack intensity, i.e., the number of attacked edge servers from 2 to 10 in steps of 2, to investigate the performance of EdgeShield under more powerful edge DDoS attacks. The results answer *Q3*. As shown in Figure 9(a), EdgeShield significantly outperforms the other five comparison systems again in terms of system throughput. The average system throughput of EdgeShield is 27.56% higher than EdgeShield$_{-ATS}$, 121.35% higher than EDMGame, 136.35% higher than PFO, 159.64% higher than SecEG, and 189.85% higher than HWS. To evaluate the robustness of EdgeShield against such attacks, Figure 9(b) compares the system throughput when more than two edge servers are under attack versus when two edge servers are under attack. When the attack intensity increases from 2 to 8, the volume of attack jobs decreases correspondingly. Some of the extra attack jobs will be processed by edge servers with available resources, which decreases the system throughput. This is consistent with Figure 9(a).

5.6 System Overhead

System overhead is an important metric to evaluate the performance of edge DDoS mitigation. System overhead

TABLE 3
System overheads (in milliseconds) under SINUS attack.

System	System Size			Edge Density			System Robustness		
	OPS	MR	$\sum OPS$	OPS	MR	$\sum OPS$	OPS	MR	$\sum OPS$
EdgeShield	50	464	23,204	45	432	19,463	58	417	24,051
EdgeShield$_{-ATS}$	126	100	12,587	135	100	13,513	161	100	16,084
EDMGame	4,584	100	458,393	5,738	100	573,750	6,693	100	669,329
PFO	127	100	12,712	143	100	14,269	154	100	15,383
SecEG	93	100	9,931	79	100	7,910	106	100	10,631
HWS	145	100	14,523	48	127	12,738	184	100	18,417

The lowest values in each column are highlighted in grey. OPS (Overhead per Strategy) is the average time taken to formulate a mitigation strategy. MR is the number of mitigation runs. $\sum OPS$ is the overall time taken to formulate all the mitigation strategies.

is measured by the time taken to formulate mitigation strategies. A system that takes a long time to formulate a mitigation strategy is impractical for mitigating edge DDoS attacks in real-world scenarios. In particular, if a system freezes in one of the time slots, the delay caused will cascade to subsequent time slots because the jobs arriving in these time slots will also be delayed. This will increase the service delay profoundly.

Table 3 summarizes the average system overheads in experiments for evaluating system scalability (§5.4) and system robustness (§5.5), and answers ***Q4***.. EdgeShield takes the least time to formulate an individual mitigation strategy. EDMGame and EdgeShield$_{-ATS}$ take the most time and second most time. We investigated and found the reason. When a lot of jobs arrive during an edge DDoS attack, EdgeShield employs ATS to reduce the time slot length. The jobs will be migrated and processed in small batches, making it easier for AME to formulate mitigation strategies, one for each batch. This lowers EdgeShield's system overhead in a single time slot. However, EdgeShield$_{-ATS}$ does not have the support of ATS, and formulates mitigation strategies for large job batches, which take more time than small job batches tremendously. As a result, EdgeShield formulates many more mitigation strategies over the same period of time to mitigate edge DDoS attacks in a fine-grained manner. EDMGame applies the potential game theory in its algorithm and requires hundreds of rounds with information communication and exchange among edge servers. Thus, communication time contributes a significant component to the overhead. SecEG incurs the least system overheads. This comes from the absence of collaborations among edge servers. However, its absence of collaborations is also the reason why SecEG achieves the second poorest performance in terms of system throughput and service delay. Overall, EdgeShield incurs the least system overheads in formulating an individual mitigation strategy, taking 63.74%, 99.10%, 63.92%, 44.96%, and 59.42% less time than EdgeShield$_{-ATS}$, EDMGame, PFO, SecEG, and HWS, respectively. This demonstrates the low system overhead of EdgeShield.

6 Conclusion and Future Work

In this paper, we proposed EdgeShield, a novel system that mitigates edge DDoS attacks without the need for attack detection. By adjusting the time slot length adaptively and triggering strategy adjustments on demand, EdgeShield can leverage edge servers' resources collectively to mitigate edge DDoS attacks effectively and efficiently. It outperforms all benchmark systems significantly with a 27.17%-78.55% system throughput gain over the state-of-the-art solution and a 58.12%-66.00% service delay reduction with acceptable system overheads. As part of future work, we will investigate the potential vulnerabilities of EdgeShield and explore corresponding defense mechanisms.

References

[1] A. G. Alcoz, M. Strohmeier, V. Lenders, and L. Vanbever, "Aggregate-based congestion control for pulse-wave DDoS defense," in *ACM SIGCOMM Conference*, 2022, pp. 693–706.

[2] P. S. Barreto, H. Y. Kim, B. Lynn, and M. Scott, "Efficient algorithms for pairing-based cryptosystems," in *Annual international cryptology conference*. Springer, 2002, pp. 354–369.

[3] A. Bremler-Barr, E. Brosh, and M. Sides, "DDoS attack on cloud auto-scaling mechanisms," in *IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[4] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *Annual International Cryptology Conference*. Springer, 2001, pp. 524–541.

[5] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using ARIMA model and its impact on cloud applications' QoS," *IEEE transactions on cloud computing*, vol. 3, no. 4, pp. 449–458, 2014.

[6] H. Chen, Y. Chen, D. H. Summerville, and Z. Su, "An optimized design of reconfigurable PSD accelerator for online shrew DDoS attacks detection," in *2013 Proceedings IEEE INFOCOM*. IEEE, 2013, pp. 1780–1787.

[7] J. Chen and X. Ran, "Deep learning with edge computing: A review." *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[8] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, 2018.

[9] C. Cimpanu, "You can now rent a Mirai botnet of 400,000 bots," http://www.bleepingcomputer.com/news/security/you-can-now-rent-a-mirai-botnet-of-400-000-bots/, 24 Nov. 2016.

[10] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proceedings of the ACM symposium on Cloud computing*, 2010, pp. 143–154.

[11] H. M. Demoulin, I. Pedisich, N. Vasilakis, V. Liu, B. T. Loo, and L. T. X. Phan, "Detecting asymmetric application-layer denial-of-service attacks in-flight with finelame," in *2019 USENIX Annual Technical Conference*, 2019, pp. 693–708.

[12] Y. Ding, K. Li, C. Liu, and K. Li, "A potential game theoretic approach to computation offloading strategy optimization in end-edge-cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1503–1519, 2021.

[13] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and elastic DDoS defense," in *USENIX Security Symposium*, 2015, pp. 817–832.

[14] P. K. Gadepalli, G. Peach, L. Cherkasova, R. Aitken, and G. Parmer, "Challenges and opportunities for efficient serverless computing at the edge," in *38th Symposium on Reliable Distributed Systems*. IEEE, 2019, pp. 261–2615.

[15] N. Garg, M. Sellathurai, V. Bhatia, B. Bharath, and T. Ratnarajah, "Online content popularity prediction and learning in wireless edge caching," *IEEE Transactions on Communications*, vol. 68, no. 2, pp. 1087–1100, 2019.

[16] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Symposium on Operating Systems Principles*, 2017, pp. 51–68.

[17] G. Gordon and R. Tibshirani, "Karush-kuhn-tucker conditions," *Optimization*, vol. 10, no. 725/36, p. 725, 2012.

[18] Q. He, C. Wang, G. Cui, B. Li, R. Zhou, Q. Zhou, Y. Xiang, H. Jin, and Y. Yang, "A game-theoretical approach for mitigating edge DDoS attack," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 4, pp. 2333 – 2348, 2022.

[19] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. Zhang, "Characterization and prediction of deep learning workloads in large-scale GPU datacenters," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–15.

[20] H. Huang, T. Meng, J. Guo, X. Wei, and W. Jia, "Seceg: A secure and efficient strategy against ddos attacks in mobile edge computing," *ACM Transactions on Sensor Networks*, vol. 20, no. 3, pp. 1–21, 2024.

[21] Y. Huang, Y. Zeng, F. Ye, and Y. Yang, "Fair and protected profit sharing for data trading in pervasive edge computing environments," in *IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1718–1727.

[22] B. Li, Q. He, F. Chen, L. Lyu, A. Bouguettaya, and Y. Yang, "Edgedis: Enabling fast, economical, and reliable data dissemination for mobile edge computing," *IEEE Transactions on Services Computing*, 2023.

[23] Y. Li, Y. Chen, T. Lan, and G. Venkataramani, "Mobiqor: Pushing the envelope of mobile edge computing via quality-of-result optimization," in *2017 IEEE 37th International Conference on Distributed Computing Systems*. IEEE, 2017, pp. 1261–1270.

[24] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 751–766.

[25] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu, "Characterizing microservice dependency and performance: Alibaba trace analysis," in *Proceedings of the ACM Symposium on Cloud Computing*, 2021, pp. 412–426.

[26] X. Ma, A. Zhou, S. Zhang, and S. Wang, "Cooperative service caching and workload scheduling in mobile edge computing," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2076–2085.

[27] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *40th Annual Symposium on Foundations of Computer Science*. IEEE, 1999, pp. 120–130.

[28] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference*, 2014, pp. 305–319.

[29] H. Park, S. Zhai, L. Lu, and F. X. Lin, "StreamBox-TZ: secure stream analytics at the edge with TrustZone," in *USENIX Annual Technical Conference*, 2019, pp. 537–554.

[30] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *IEEE Conference on Computer Communications*. IEEE, 2019, pp. 514–522.

[31] Q. Pei, S. Chen, Q. Zhang, X. Zhu, F. Liu, Z. Jia, Y. Wang, and Y. Yuan, "Cooledge: hotspot-relievable warm water cooling for energy-efficient edge datacenters," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 814–829.

[32] J. Peng, Q. Li, X. Ma, Y. Jiang, Y. Dong, C. Hu, and M. Chen, "Magnet: Cooperative edge caching by automatic content congregating," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 3280–3288.

[33] S. Ranjan, R. Swaminathan, M. Uysal, A. Nucci, and E. Knightly, "DDoS-shield: DDoS-resilient scheduling to counter application layer attacks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 1, pp. 26–39, 2008.

[34] A. Scherrer, N. Larrieu, P. Owezarski, P. Borgnat, and P. Abry, "Non-gaussian and long memory statistical characterizations for internet traffic with anomalies," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 1, pp. 56–70, 2007.

[35] M. Shahrad, R. Fonseca, Í. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *USENIX Annual Technical Conference*, 2020, pp. 205–218.

[36] G. Somani, M. S. Gaur, D. Sanghi, M. Conti, and R. Buyya, "DDoS attacks in cloud computing: Issues, taxonomy, and future directions," *Computer Communications*, vol. 107, pp. 30–48, 2017.

[37] G. Somani, M. S. Gaur, D. Sanghi, M. Conti, and M. Rajarajan, "Scale inside-out: Rapid mitigation of cloud DDoS attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 959–973, 2017.

[38] C. W. Tan, D.-M. Chiu, J. C. Lui, and D. K. Yau, "A distributed throttling approach for handling high bandwidth aggregates," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 7, pp. 983–995, 2007.

[39] Z. Tao, Q. Xia, Z. Hao, C. Li, L. Ma, S. Yi, and Q. Li, "A survey of virtual machine management in edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1482–1499, 2019.

[40] M. Tran, M. S. Kang, H.-C. Hsiao, W.-H. Chiang, S.-P. Tung, and Y.-S. Wang, "On the feasibility of rerouting-based DDoS defenses," in *IEEE Symposium on Security and Privacy*. IEEE, 2019, pp. 1169–1184.

[41] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, "DDoS attack protection in the era of cloud computing and software-defined networking," *Computer Networks*, vol. 81, pp. 308–319, 2015.

[42] K. Wang, Q. He, F. Chen, C. Chen, F. Huang, H. Jin, and Y. Yang, "Flexifed: Personalized federated learning for edge clients with heterogeneous model architectures," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 2979–2990.

[43] S. Wu, X. Wang, H. Jin, and H. Chen, "Elastic resource provisioning for batched stream processing system in container cloud," in *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data*. Springer, 2017, pp. 411–426.

[44] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Online collaborative data caching in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 281–294, 2020.

[45] Y. Xiang, K. Li, and W. Zhou, "Low-rate DDoS attacks detection and traceback by using new information metrics," *IEEE transactions on information forensics and security*, vol. 6, no. 2, pp. 426–437, 2011.

[46] L. Yuan, Q. He, S. Tan, B. Li, J. Yu, F. Chen, H. Jin, and Y. Yang, "CoopEdge: A decentralized blockchain-based platform for cooperative edge computing," in *Proceedings of the Web Conference*, 2021, pp. 2245–2257.

[47] C. Zeng, L. Luo, T. Zhang, Z. Wang, L. Li, W. Han, N. Chen, L. Wan, L. Liu, Z. Ding *et al.*, "Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing," in *19th USENIX Symposium on Networked Systems Design and Implementation*, 2022, pp. 1345–1358.

[48] R. Zhou, Y. Zeng, L. Jiao, Y. Zhong, and L. Song, "Online and predictive coordinated cloud-edge scrubbing for ddos mitigation," *IEEE Transactions on Mobile Computing*, 2024.

[49] L. Zhu, X. Tang, M. Shen, X. Du, and M. Guizani, "Privacy-preserving ddos attack detection using cross-domain traffic in software defined networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 628–643, 2018.

APPENDIX A
BMF AND VRF PROPERTIES

Before presenting the election process in detail, we introduce the BMF and VRF:

- **BMF.** EdgeShield employs the BMF to verify the identities of the participants in the election process. It can be represented with $e(\mathbb{L}_1, \mathbb{L}_1) \rightarrow \mathbb{L}_2$, where $\mathbb{L}_1$ and $\mathbb{L}_2$ are multiplicative cyclic lists with a large prime order p. The generator of $\mathbb{L}_1$ is denoted with δ. BMF is non-degenerate, i.e., $e(\sigma, \tau) \neq 1, \forall \sigma, \tau \in \mathbb{L}_1$. In addition, BMF fulfills $e(\sigma^{\varsigma}, \tau^{\epsilon}) = e(\sigma^{\epsilon}, \tau^{\varsigma}) = e(\sigma^{\varsigma\epsilon}, \tau) = e(\sigma, \tau^{\varsigma\epsilon}) = e(\sigma, \tau)^{\varsigma\epsilon}, \forall \varsigma, \epsilon \in \mathbb{Z}_p^*$.
- **VRF.** Edge servers run the VRF to generate their bids for leadership. Given a random seed $seed$[3], an edge server runs the VRF to generate a private key key_{pri}, a public key key_{pub}, a tag $tag_{key_{pri}}(seed) = e(\delta, \delta)^{1/(seed+key_{pri})}$ and a random VRF value $value_{key_{pri}}(seed) = \delta^{1/(seed+key_{pri})}$. For $value_{key_{pri}}(seed)$ to be valid, $e(\delta^{seed} \times key_{pub}, tag_{key_{pri}}(seed)) = e(\delta, \delta)$ and $e(\delta, tag_{key_{pri}}(seed)) = value_{key_{pri}}(seed)$ must hold.

3. Many approaches have been developed to generate and distribute random seed values. EdgeShield adopts the one proposed in [4], which is also adopted by Algorand [16].