CARBINE: Exploring Additional Properties of HyperLogLog for Secure and Robust Flow Cardinality Estimation

Damu Ding
University of Oxford, UK & Bytedance Inc., USA
damu.ding@eng.ox.ac.uk

***Abstract*—Counting distinct elements (also named flow cardinality) of large data streams in the network is of primary importance since it can be used for many practical monitoring applications, including DDoS attack and malware spread detection. However, modern intrusion detection systems are struggling to reduce both memory and computational overhead for such measurements. Many algorithms are designed to estimate flow cardinality, in which HyperLogLog has been proven the most efficient due to its high accuracy and low memory usage. While HyperLogLog provides good performance on flow cardinality estimation, it has inherent algorithmic vulnerabilities that lead to both security and robustness issues. To overcome these issues, we first investigate two possible threats in HyperLogLog, and propose corresponding detection and protection solutions. Leveraging proposed solutions, we introduce CARBINE, an approach that aims at identifying and eliminating the threats that most probably mislead the output of HyperLogLog. We implement our CARBINE to evaluate the threat detection performance, especially in case of a practical network scenario under volumetric DDoS attack. The results show that our CARBINE can effectively detect different kinds of threats while performing even higher accuracy and update speed than original HyperLogLog.**

I. Introduction

Network monitoring [1]–[8] plays a key role in network management as it can collect network data and infer statistics to diagnose network performance and security issues [9]. However, the monitoring in modern intrusion detection systems is struggling to measure traffic in high-speed networks as the network data becomes distributed and massive. Collecting exact measurement of extensive network data streams is often impractical due to excessively high memory occupation and computational/communication overhead, and utilizing probabilistic estimation through compact data structures (e.g. sketches) is a feasible alternative.

In network monitoring, flow cardinality estimation [10], [11] is a fundamental problem, which estimates the number of distinct flows in a predefined measurement time interval. Each flow is uniquely identified by one or multiple fields in the packet headers, called *flow key*, which can be flexibly defined based on monitoring application requirements. The flow key under measurement can be any subset of 5 tuple or other fields that appear in packet header. For example, for each destination, if source IPs are treated as flow key, then the flow cardinality indicates the number of distinct source addresses that contact the same destination, which can be used as a metric for DDoS victim identification [12]. Existing research on flow cardinality estimation mainly focuses on the design of sketches to store the summary of raw traffic data within small amounts of memory. As illustrated in Fig.1, the sketches are usually offloaded to a monitoring server for measuring traffic and local query, with the goal to minimize the communication overhead between collector and servers. On the other side, the collector

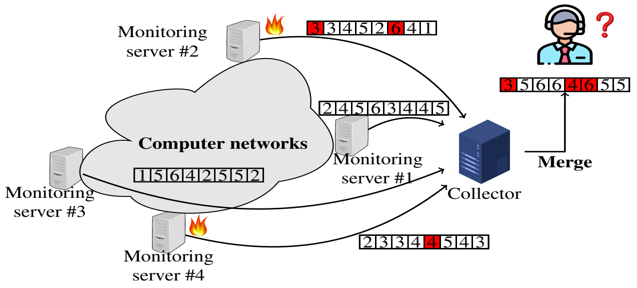


Fig. 1: Deployment scenario

is responsible for merging sketches from multiple servers and monitoring global network status. However, if no protection of sketches is considered, the values in the sketches may be modified by attackers, leading to large bias on the estimations in both servers and collectors. The inaccurate estimations may further cause wrong inference on anomaly detection.

Therefore, the goal of this paper is to study the security and robustness of HyperLogLog [13], which is the most common and practical sketch used for flow cardinality estimation in network monitoring [14]–[16]. Comparing to existing sketch-based flow cardinality estimation algorithms, HyperLogLog is more efficient on the estimation of large data streams due to its high accuracy and low memory usage. Furthermore, multiple HyperLogLog sketches can be easily merged into a single sketch to compute the union of different packet streams, which is appropriate for the deployment scenario in Fig.1.

As HyperLogLog is widely used in monitoring for anomaly detection purpose (e.g. DDoS [10] and port scan detection [17]), attackers can try to evade the detection or cause false alarms by exploiting the vulnerabilities of HyperLogLog. For instance, the attackers can handle the hashed inputs of HyperLogLog by varying the flow key to prevent the increments of estimated flow cardinality [18]. Similarly, the attackers can also inflate the values in HyperLogLog with a set of flows containing abnormal large values to cause detection false positives.

With those threats in mind, we present a theoretical analysis of HyperLogLog to explore its additional properties that can be used to detect aforementioned threats. We then propose corresponding detection and protection solutions against the threats. Based on proposed solutions, we present CARBINE (seCure And RoBust Improved caridNality Estimation), an approach aims to enhance the security and robustness of HyperLogLog and provide accurate flow cardinlaity estimation in monitoring servers. The evaluation results on real network flow traces show that CARBINE is effective to detect two different kinds of threats while maintaining good accuracy and high update speed. In summary, we make the following contributions:

- We theoretically analyze HyperLogLog and explore its additional properties, including mode, maximum and minimum value. We then show how we can derive them from the sum of register values. Additionally, we also find some new relations of packet count in HyperLogLog that can be used for the detection of threats.
- Based on our theoretical analysis, we present CARBINE, an improved HyperLogLog approach for secure and robust flow cardinality estimation in monitoring servers. CARBINE is able to detect two different kinds of threat models and protect HyperLogLog.
- We evaluate our approach using real-world Internet traces, and show good accuracy and high speed on threats detection. In particular, we also give a case study considering volumetric DDoS attacks within a practical network scenario.

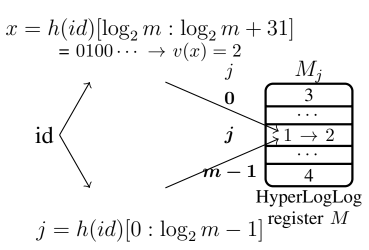
II. Background of HyperLogLog

In this section, we first introduce HyperLogLog algorithms in detail. We then present additional properties of HyperLogLog through theoretical analysis.

A. Basic idea of HyperLogLog

HyperLogLog [13] is a sketch-based algorithm that can be used to estimate large number of distinct flows traversing a network monitoring point. They envision two types of operations: *Update* and *Query*. Update operation updates the HyperLogLog sketch with flow information from the incoming packet, whereas Query operation is adopted to retrieve from the sketch the estimated flow cardinality. As shown in Fig. 2, the Update operation works as follows: given an incoming packet with *flow key id* (e.g. any subset of 5 tuple) and an m-sized ($m \in \{2^4, 2^5, \cdots, 2^{16}\}$) HyperLogLog register M with l bits each cell, HyperLogLog applies to id a uniform distributed hash function h with output size os ($os \geq 2^l + \log_2 m$): the resulted os-bit binary string $h(id)$ is denoted by $h(id) = [0 : os-1]$. HyperLogLog then updates an m-sized register M. Let j be the leftmost $\log_2 m$ bits of $h(id)$ and x the 2^l bits of remaining, i.e., $j = h(id)[0 : \log_2 m - 1]$ and $x = h(id)[\log_2 m : \log_2 m + 2^l - 1]$. M is updated following this rule: $M_j = max(M_j, v(x))$, where $v(x)$ is the index of the leftmost 1 of x plus one. For instance, x is 0100, then $v(x)$ is 2. Fig. 3 shows the probability of $v(x)$ in HyperLogLog: the possible appearance of $v(x) = v$ is 2^{-v} of all incoming packets. Register M can then be queried to estimate the flow cardinality $\hat{n}_{tot}$ when $\hat{n}_{tot}$ is within the range $(\frac{5}{2}m, \frac{1}{30}2^{32}]$ using *Harmonic mean* of power of 2, that is, $\hat{n}_{tot} = \alpha_m^{HLL} m^2 (\sum_0^{m-1} 2^{-M_j})^{-1}$, in which α_m^{HLL} is a bias correction parameter of HyperLogLog. In this paper, we will only focus on the flow cardinality estimation in this range. For the estimation beyond this range, please refer to the original HyperLogLog [13] for small and large range corrections.

Another algorithm called *LogLog* [19] performs the same Update operation as HyperLogLog, but LogLog uses *arithmetic mean* to estimate the flow cardinality $\hat{n}_{tot}$, which is computed as $\hat{n}_{tot} = \alpha_m m 2^{\frac{Sum}{m}}$, where α_m is a bias correction parameter of LogLog and Sum equals to $\sum_{j=0}^{m-1} M_j$. The


$x = h(id)[\log_2 m : \log_2 m + 31]$
$= 0100\cdots \rightarrow v(x) = 2$
j
M_j
id
0
j
$m-1$
3
$\cdots$
$1 \rightarrow 2$
$\cdots$
4
HyperLogLog register M
$j = h(id)[0 : \log_2 m - 1]$
Fig. 2: Scheme of HyperLogLog updates

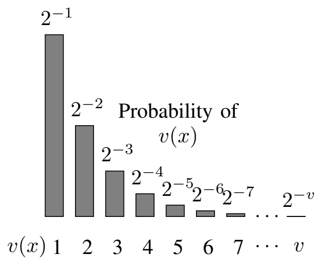


Fig. 3: Probability of $v(x)$ entering in HyperLogLog

theoretical standard error of LogLog is $1.30/\sqrt{m}$, and that of HyperLogLog is $1.04/\sqrt{m}$, where m is the size of register M. Note that LogLog is required in the following to explore new properties of HyperLogLog.

An interesting property of HyperLogLog is that multiple same-sized HyperLogLog sketches can be merged into a single sketch, which can be used to count the flow cardinality of the union of many packet streams. Note that all sketches should use the same hash functions. Considering b HyperLogLog sketches $M^1, M^2, \cdots, M^b$ with size m, the merged HyperLogLog sketch satisfies $M^{merge} = [max(M^1_{j=0}, M^2_{j=0}, \cdots, M^b_{j=0}), \cdots, max(M^1_{j=m-1}, M^2_{j=m-1}, \cdots, M^p_{j=m-1})]$. Finally, the merged sketch can be queried by the same way as a single HyperLogLog.

B. Additional properties of HyperLogLog

In this section, we present a theoretical analysis to explore additional properties of HyperLogLog, including mode (i.e. the most frequent value), maximum and minimum value in m-sized HyperLogLog register M. If not otherwise specified, the register cell size l_{cell} is 5 bits each, i.e., the value in each register cell is up to 31. We use symbol n_{tot} to denote the actual flow cardinality, and $\hat{n}_{tot}$ its estimation.

1) Mode, minimum and maximum in HyperLogLog register:

Theorem 1. *As $n_{tot} \rightarrow \infty$, if k_{mode} is the mode of the HyperLogLog register M, $e^{-\frac{n_{tot}}{m2^{k_{mode}}}} \rightarrow 0.5$*

Proof. According to [13], the values M_j in HyperLogLog register satisfy following probability distribution:

$$\mathbb{P}(M_j = k) = (1 - \frac{1}{m2^k})^{n_{tot}} - (1 - \frac{1}{m2^{k-1}})^{n_{tot}}$$

where k is the value in M_j.

As $n_{tot} \rightarrow \infty$, the equation can be converted to the following based on $(1+x)^\alpha \approx e^{\alpha x}$ when $|x|$ is small and $|\alpha x|$ is large:

$$\mathbb{P}(M_j = k) = e^{-\frac{n_{tot}}{m2^k}} - e^{-\frac{2n_{tot}}{m2^k}} = -(e^{-\frac{n_{tot}}{m2^k}} - 0.5)^2 + 0.25$$

It shows that when $e^{-\frac{n_{tot}}{m2^k}} \rightarrow 0.5$, $\mathbb{P}(M_j = k)$ reaches the maximum. In this case, $k = k_{mode}$, that is, k_{mode} is the most frequent value appearing in HyperLogLog register M_j ($0 \leq j \leq m-1$). ☐

Theorem 2. *With a probability at least $\frac{m-2}{m-1}$, the minimum value in the HyperLogLog register is $k_{min} = k_{mode} - \lfloor \log_2 \log_2 m \rfloor$ if $k_{mode} > \lfloor \log_2 \log_2 m \rfloor$ ($\lfloor\rfloor$ is the floor function), otherwise, the minimum value is 0.*

Proof. If $k = k_{mode} - \xi(\textit{integer } \xi \geq 1)$, the probability is:

$$\begin{aligned}\mathbb{P}(M_j = k_{mode} - \xi) &= -(e^{-\frac{n_{tot}}{m2^{k_{mode}-\xi}}} - 0.5)^2 + 0.25\\ &= -((e^{-\frac{n_{tot}}{m2^{k_{mode}}}})^{2^{\xi}} - 0.5)^2 + 0.25\\ \textit{(By Theorem 1)} &\to -((0.5)^{2^{\xi}} - 0.5)^2 + 0.25\\ &= 0.5^{2^{\xi}} - (0.5^{2^{\xi}})^2 = 0.5^{2^{\xi}}(1 - 0.5^{2^{\xi}})\end{aligned}$$

We use $0.5^{2^{\xi}}$ to approximate $\mathbb{P}(M_j = k_{mode} - \xi)$ because the difference between $0.5^{2^{\xi}}$ and $0.5^{2^{\xi}}(1-0.5^{2^{\xi}})$ is $0.5^{2^{\xi+1}}$, which decreases as ξ increases. When ξ is greater than 1, they are almost overlapped.

Considering the HyperLogLog register size m, and theoretically, the minimum value of the register $k_{min} = k_{mode} - \xi$ if $\mathbb{P}(k = k_{min}) \geq \frac{1}{m}$ and $\mathbb{P}(k = k_{min} - 1) < \frac{1}{m}$, meaning that no value smaller than k_{min} appears in m-sized HyperLogLog. Therefore, we have:

$$\mathbb{P}(k = k_{min}) = 0.5^{2^{\xi}} \geq \frac{1}{m} \Rightarrow m \geq 2^{2^{\xi}} \Rightarrow \xi \leq \log_2 \log_2 m$$

When $k = k_{min} - 1$, $\mathbb{P}(k = k_{min} - 1) = 0.5^{2^{\xi+1}} < \frac{1}{m}$ implies $m < 2^{2^{\xi+1}}$, and $\xi > \log_2 \log_2 m - 1$. Hence, the unique possible integer of ξ is $\lfloor \log_2 \log_2 m \rfloor$.

The probability $\mathbb{P}(k < k_{min})$ is: $\mathbb{P}(k < k_{min}) = 0.5^{2^{\xi+1}} + 0.5^{2^{\xi+2}} + 0.5^{2^{\xi+3}} + \cdots < 0.5^{2^{\xi+1}} + (0.5^{2^{\xi+1}})^2 + (0.5^{2^{\xi+1}})^3 + (0.5^{2^{\xi+1}})^4 + \cdots \to 0.5^{2^{\xi+1}}(\frac{1}{1-0.5^{2^{\xi+1}}})$. Since $0.5^{2^{\xi+1}}(\frac{1}{1-0.5^{2^{\xi+1}}})$ is a monotonic function and $0.5^{2^{\xi+1}} < \frac{1}{m}$, $\mathbb{P}(k \leq k_{min} - 1) < \frac{1}{m} \cdot \frac{1}{1-\frac{1}{m}} = \frac{1}{m-1}$.

Therefore, with a probability at least $1 - \frac{1}{m-1} = \frac{m-2}{m-1}$, k_{min} can be formulated as follows:

$$k_{min} = \begin{cases} 0 & \text{if } k_{mode} \leq \lfloor \log_2 \log_2 m \rfloor \\ k_{mode} - \lfloor \log_2 \log_2 m \rfloor & \text{otherwise} \quad \square \end{cases}$$

Theorem 3. *With a probability at least $\frac{m-1}{m}$, the maximum value in the HyperLogLog register M is $k_{max} = k_{mode} + \log_2 m - 1$ if $k_{mode} \leq 32 - \log_2 m$. Else, the maximum value is 31.*

Proof. Theorem 1 indicates that $\mathbb{P}(M_j = k_{mode}) \to 0.25$ and $e^{-\frac{n_{tot}}{m2^{k_{mode}}}} \to 0.5$. If k is larger than k_{mode} and $k = k_{mode} + \eta(\textit{integer } \eta \geq 1)$, the probability of $M_j = k_{mode} + \eta$ is:

$$\begin{aligned}\mathbb{P}(M_j = k_{mode} + \eta) &= -(e^{-\frac{2^{-\eta}n_{tot}}{m2^{k_{mode}}}} - 0.5)^2 + 0.25\\ &= -(0.5^{2^{-\eta}} - 0.5)^2 + 0.25\end{aligned}$$

When η tends to large, the equation can be further deduced by applying twice of *Binomial approximation (BA), i.e.,* $(1 - x)^{\alpha} \approx 1 - \alpha x$ *when* $|x| < 1$ *and* $|\alpha x| \ll 1$:

$$\begin{aligned}\mathbb{P}(M_j = k_{mode} + \eta) &= -((1 - 0.5)^{2^{-\eta}} - 0.5)^2 + 0.25\\ \textit{(By BA)} &= -((1 - 0.5 \cdot 2^{-\eta}) - 0.5)^2 + 0.25\\ &= -0.25(1 - 2^{-\eta})^2 + 0.25\\ \textit{(By BA)} &= -0.25(1 - 2 \cdot 2^{-\eta}) + 0.25 = 2^{-\eta-1}\end{aligned}$$

When η is the largest integer that satisfies $\mathbb{P}(k_{mode} + \eta) \geq \frac{1}{m}$, $k_{mode} + \eta$ is the maximum value:

$$\mathbb{P}(k = k_{mode} + \eta) = 2^{-\eta-1} \geq \frac{1}{m} \Rightarrow \eta \leq \log_2 m - 1$$

However, note that if $\eta' = \eta + u(u \in \mathbb{N}^+)$, the probability $\mathbb{P}(k = k_{mode} + \eta')$ is $\mathbb{P}(k = k_{mode} + \eta') = 2^{-(\log_2(m)-1+u)-1} = \frac{2^{-u}}{m}$, and $\mathbb{P}(k > k_{mode} + \eta)$ tends to $\frac{1}{m}$ because $\mathbb{P}(k > k_{mode} + \eta) = \frac{1}{m}\sum_{u=1}^{32-\log_2(m)-k_{mode}} 2^{-u} \to \frac{1}{m}\frac{2^{-1}}{1-2^{-1}} = \frac{1}{m}$. This means that $k_{mode} + \eta'$ is possible to be the largest value k_{max} in the register especially when k_{mode} is small.

Since the limit of each register cell in HyperLogLog is 31, by replacing $k_{mode} + \log_2 m - 1$ to k^{base}_{max} (named base maximum), with a probability at least $1 - \frac{1}{m} = \frac{m-1}{m}$, k_{max} is equal to:

$$k_{max} = \begin{cases} k^{base}_{max} & k_{mode} \leq 32 - \log_2 m \\ 31 & \text{otherwise} \quad \square \end{cases}$$

Remark. *Unlike mode and minimum in HyperLogLog, if there is a value greater than k^{base}_{max}, the maximum is changed. Therefore, with a probability at most $\frac{1}{m}$, the maximum k_{max} is $k^{base}_{max} + u(u \in \mathbb{N}^+)$. The parameter u depends on the hash function since different hash functions generate different largest hashed values $v(x)$.*

2) *Derive minimum and base maximum value from the sum of HyperLogLog register:*

Once understanding that the minimum and base maximum value of HyperLogLog register depend on the mode and register size m, in this subsection, we demonstrate that it is possible to directly retrieve these additional properties from the sum of HyperLogLog register. The demonstration relies on the property that LogLog uses the sum to query the flow cardinality, and note the difference of LogLog and HyperLogLog.

Lemma 1. *$\frac{n_{tot}}{m}$ is in the range $2^{k_{mode}-1} \leq \frac{n_{tot}}{m} < 2^{k_{mode}}$*

Proof. $e^{-\frac{n_{tot}}{m2^{k_{mode}}}} \to 0.5$ (By Theorem 1) implies that $\frac{n_{tot}}{m} \to 2^{k_{mode}} \ln 2$. Since $\ln 2 \approx 0.693$ and $e^{-\frac{n_{tot}}{m2^{k_{mode}}}}$ cannot be 1, $\frac{n_{tot}}{m}$ should locate within $[2^{k_{mode}-1}, 2^{k_{mode}})$. $\square$

Lemma 2. *As $n_{tot} \to \infty$, by expressing $\sum_{j=0}^{m-1} M_j$ as Sum, $\alpha_m 2^{\frac{Sum}{m}}$ approximately follows a normal distribution $\mathcal{N}(\frac{n_{tot}}{m}, (\frac{1.30}{m})^2(\frac{n_{tot}}{m})^2)$, where α_m is the correction parameter of LogLog algorithm and M is HyperLogLog register.*

Proof. As $n_{tot} \to \infty$, LogLog estimation approximately follows the normal distribution $\hat{n}_{LL} = \alpha_m m 2^{\frac{Sum}{m}} \sim \mathcal{N}(\mu_{LL}, \sigma^2_{LL})$, where $\mu_{LL} = n_{tot}$ and $\sigma^2_{LL} = (\frac{1.30}{\sqrt{m}})^2 n^2_{tot}$. A similar approach has been proven in [18]. According to the linearity of normal distribution, $\frac{\hat{n}_{LL}}{m} = \alpha_m 2^{\frac{Sum}{m}} \sim \mathcal{N}(\frac{1}{m}\mu_{LL}, \frac{1}{m^2}\sigma^2_{LL})$. Applying $\mu_p = \frac{1}{m}\mu_{LL} = \frac{n_{tot}}{m}$ and $\sigma^2_p = \frac{1}{m^2}\sigma^2_{LL} = (\frac{1.30}{\sqrt{m}})^2(\frac{n_{tot}}{m})^2$, $\alpha_m 2^{\frac{Sum}{m}}$ approximately follows a normal distribution $\mathcal{N}(\mu_p, \sigma^2_p)$, that is, $\alpha_m 2^{\frac{Sum}{m}}$ is an asymptotically unbiased estimator of $\frac{n_{tot}}{m}$ with coefficient of variation $\frac{\sigma_p}{\mu_p} = \frac{\frac{1.30}{\sqrt{m}}\frac{n_{tot}}{m}}{\frac{n_{tot}}{m}} = \frac{1.30}{\sqrt{m}}$. $\square$

Lemma 3. *As $n_{tot} \to \infty$ and $m \to \infty$, given α_m is the correction parameter of LogLog algorithm, $\log_2(\alpha_m 2^{\frac{Sum}{m}})$ is an unbiased estimator of $\log_2(\frac{n_{tot}}{m})$, i.e. $\log_2(\alpha_m 2^{\frac{Sum}{m}}) = \log_2(\frac{n_{tot}}{m})$.*

Proof. By Lemma 2, $\frac{\sigma_p}{\mu_p} = \frac{1.30}{\sqrt{m}}$, as m is usually very large, $\frac{\sigma_p}{\mu_p} = \frac{1.30}{\sqrt{m}} \to 0$. This allows us to use *Taylor expansions* to accurately approximate the expectation and variance of $\log_2 \alpha_m 2^{\frac{Sum}{m}}$: $\mathbb{E}[\log_2 \alpha_m 2^{\frac{Sum}{m}}] \approx \log_2 \mu_p + \frac{(\log_2 \mu_p)''}{2}\sigma_p^2 = \log_2 \mu_p - \frac{\sigma_p^2}{2\ln 2 \cdot \mu_p^2}$ and $Var[\log_2 \alpha_m 2^{\frac{Sum}{m}}] \approx ((\log_2 \mu_p)')^2 \sigma_p^2 = \frac{\sigma_p^2}{(\ln 2 \mu_p)^2}$. As $\frac{\sigma_p}{\mu_p} \to 0$, $\mathbb{E}[\log_2 \alpha_m 2^{\frac{Sum}{m}}] \to \log_2 \mu_p = \log_2(\frac{n_{tot}}{m})$ and $Var[\log_2 \alpha_m 2^{\frac{Sum}{m}}] \to 0$. □

Lemma 4. *Being α_m the correction parameter of LogLog, the mode k_{mode} in HyperLogLog can be estimated as $\lceil \log_2 \alpha_m 2^{\frac{Sum}{m}} \rceil$ ($\lceil\rceil$ is the ceil function).*

Proof. By Lemma 1, k_{mode} should satisfy $k_{mode} = \lceil \log_2 \frac{n_{tot}}{m} \rceil$. By Lemma 2 and 3, we can approximate $\frac{n_{tot}}{m}$ with $\alpha_m 2^{\frac{Sum}{m}}$, and so: $k_{mode} = \lceil \log_2 \alpha_m 2^{\frac{Sum}{m}} \rceil$ □

Theorem 4. *The minimum value k_{min} in HyperLogLog register can be estimated as $\lceil \log_2 \alpha_m 2^{\frac{Sum}{m}} \rceil - \lfloor \log_2 \log_2 m \rfloor$ if $Sum > T_{min} m$, where $T_{min} = (\lfloor \log_2 \log_2 m \rfloor + 1.33)$.*

Proof. By Theorem 2, $k_{min} = 0$ if $k_{mode} \le \lfloor \log_2 \log_2 m \rfloor$. Given α_m = 0.39701 (constant in LogLog), applying Lemma 4 implies:

$$\begin{aligned} \lceil \log_2 \alpha_m 2^{\frac{Sum}{m}} \rceil &\le \lfloor \log_2 \log_2 m \rfloor \\ \log_2 \alpha_m 2^{\frac{Sum}{m}} &\le \lfloor \log_2 \log_2 m \rfloor (Since \lfloor \log_2 \log_2 m \rfloor \in \mathbb{N}^+) \\ Sum &\le (\lfloor \log_2 \log_2 m \rfloor + 1.33)m \end{aligned}$$

and $k_{min} = k_{mode} - \lfloor \log_2 \log_2 m \rfloor = \lceil \log_2 \alpha_m 2^{\frac{Sum}{m}} \rceil - \lfloor \log_2 \log_2 m \rfloor$. More generally, by replacing $(\lfloor \log_2 \log_2 m \rfloor + 1.33)m$ to T_{min}, for any integer $p \ge 1$ ($p = \lceil \log_2 \alpha_m 2^{\frac{Sum}{m}} \rceil - \lfloor \log_2 \log_2 m \rfloor$), $k_{min} = p$ if $k_{mode} > \lfloor \log_2 \log_2 m \rfloor + p - 1$ and $k_{mode} = \lfloor \log_2 \log_2 m \rfloor + p$, which means that:

$$\begin{aligned} \lfloor \log_2 \log_2 m \rfloor + p - 1 < \lceil \log_2 \alpha_m 2^{\frac{Sum}{m}} \rceil &= \lfloor \log_2 \log_2 m \rfloor + p \\ \lfloor \log_2 \log_2 m \rfloor + p - 1 < \log_2 \alpha_m 2^{\frac{Sum}{m}} &\le \lfloor \log_2 \log_2 m \rfloor + p \\ (T_{min} + p - 1)m < Sum &\le (T_{min} + p)m \end{aligned}$$

Finally, we can express k_{min} as follows:

$$k_{min} = \begin{cases} 0 & \text{if } Sum \le T_{min} m \\ p & \text{if} (T_{min} + p - 1)m < Sum \le (T_{min} + p)m \end{cases}$$ □

Theorem 5. *The base maximum value k_{max}^{base} in HyperLogLog register can be estimated as $\lceil \log_2 \alpha_m 2^{\frac{Sum}{m}} \rceil + \log_2 m - 1$ if $Sum < (33.33 + \log_2 m)m$.*

Proof. According to Theorem 3, $k_{max}^{base} = 31$ if $k_{mode} \ge 32 - \log_2 m$. By replacing k_{mode} to $\lceil \log_2 \alpha_m 2^{\frac{Sum}{m}} \rceil (\alpha_m = 0.39701)$ as proofed in Lemma 4 yields:

$$\begin{aligned} \lceil \log_2 \alpha_m 2^{\frac{Sum}{m}} \rceil &\ge 32 - \log_2 m \\ \log_2 \alpha_m 2^{\frac{Sum}{m}} &\ge 32 - \log_2 m - 1 \\ Sum &\ge (32.33 + \log_2 m)m \end{aligned}$$

and k_{max}^{base} is:

$$k_{max}^{base} = k_{mode} + \log_2 m - 1 = \lceil \log_2 \alpha_m 2^{\frac{Sum}{m}} \rceil + \log_2 m - 1$$

Therefore, for any integer $q > k_{mode}$, when $k_{max}^{base} = q$, $k_{mode} = q + 1 - \log_2 m$, and:

$$\begin{aligned} q - \log_2 m < \lceil \log_2 \alpha_m 2^{\frac{Sum}{m}} \rceil &= q + 1 - \log_2 m \\ q - \log_2 m < \log_2 \alpha_m 2^{\frac{Sum}{m}} &\le q + 1 - \log_2 m \\ (1.33 + q - \log_2 m)m < Sum &\le (2.33 + q - \log_2 m)m \end{aligned}$$

Finally, by defining $1.33 - \log_2 m$ as T_{max}, for any $q > k_{mode}$, we can express k_{max} as follows:

$$k_{max}^{base} = \begin{cases} q & \text{if } (T_{max} + q)m \le Sum < (T_{max} + q + 1)m \\ 31 & \text{if } Sum \ge (32.33 + \log_2 m)m \end{cases}$$ □

Remark. *Theorem 4 and 5 show that k_{min} and k_{max}^{base} change only in case that Sum increments m (i.e. the size of HyperLogLog register). This means that when there are few abnormal large values in the HyperLogLog register, the estimations of k_{min} and k_{max}^{base} are not affected.*

3) The relation between the sums of different HyperLogLog registers on the same packet stream: Considering two m-sized HyperLogLog registers, namely $HLL1$ and $HLL2$, with different input hash functions to estimate the flow cardinality, the sum of $HLL1$ is $Sum1$ and that of $HLL2$ is $Sum2$. Given the actual flow cardinality is n_{tot}, the estimated flow cardinality by $HLL1$ is $\hat{n}_1$, and that of $HLL2$ is $\hat{n}_2$.

Lemma 5. *The sum of HyperLogLog register follows a normal distribution $\mathcal{N}(\mu_{Sum}, \sigma_{Sum}^2)$ where $\mu_{Sum} = m\log_2(n_{tot}) - m\log_2(\alpha_m m)$, and $\sigma_{Sum}^2 = 3.51m$*

Proof. Since LogLog estimates flow cardinality as $\hat{n}_{tot} = \alpha_m m 2^{\frac{Sum}{m}}$, $Sum = m\log_2(\hat{n}_{tot}) - m\log_2(\alpha_m m)$, the expectation is $\mathbb{E}[\hat{n}_{tot}] = n_{tot}$, and the variance $Var[\hat{n}_{tot}] = \frac{1.30^2}{m} n_{tot}^2$. Similar to what we proofed in Lemma 3, applying Taylor expansion yields: $\mathbb{E}[Sum] = m\log_2(n_{tot}) - m\log_2(\alpha_m m)$ and $Var[Sum] = m^2 Var[\log_2(n_{tot})] = m^2 \frac{Var[\hat{n}_{tot}]}{(\ln 2 \mathbb{E}[\hat{n}_{tot}])^2} = 3.51m$. □

Theorem 6. *The difference of Sum1 and Sum2 follows a normal distribution $\mathcal{N}(0, 7.02m)$, where m is the HyperLogLog register size.*

Proof. Given $Sum1 - Sum2 = m\log_2(\hat{n}_1) - m\log_2(\hat{n}_2)$, the expectation of $Sum1 - Sum2$ is 0, and the variance is $2 \cdot 3.51m = 7.02m$ □

Remark. *Possible attacks on HyperLogLog register for monitoring purpose can be detected if the sum difference of two HyperLogLog registers is greater than w times of standard deviation $\sqrt{7.02m}$, where w can be chosen by network operator based on standard normal distribution table.*

4) The relation between the packets with $v(x) = 1$ and the total number of packets: Considering a packet stream $S = \{f_1, f_2, \cdots, f_{n_{tot}}\}$, of which the total number of packets is S_{tot} and flow number is n_{tot} ($n_{tot} \le S_{tot}$). Moreover, S_{1s} represents the number of packets with $v(x) = 1$ and n_{1s} denotes its flow number:

Lemma 6. *Being $R_{1s} = \frac{S_{1s}}{n_{1s}}$ the average packet number per flow with $v(x) = 1$ and $R_{tot} = \frac{S_{tot}}{n_{tot}}$ the overall average* *flow packet number, as $n_{1s} \to \infty$, R_{1s} follows a normal distribution $\mathcal{N}(\mu, \frac{\sigma_S^2}{n_{1s}})$, where the expectation μ is equal to R_{tot} and σ_S^2 is the variance of flow packet count f_i in packet stream S (i.e. $\sigma_S^2 = \frac{\sum_{i=1}^{n_{tot}}(f_i-\mu)^2}{n_{tot}}$).*

Proof. Given the packets with $v(x) = 1$ are n_{1s} random samples from n_{tot}, according to Central limit Theorem, the mean value of n_{1s} random samples $\frac{S_{1s}}{n_{1s}}$, follows a normal distribution with expectation $R_{tot} = \frac{S_{tot}}{n_{tot}}$ and variance $\frac{\sigma_S^2}{n_{1s}}$. □

Theorem 7. *As $n_{tot} \to \infty$, $\frac{S_{1s}}{S_{tot}}$ follows a normal distribution $\mathcal{N}(0.5, \sigma_S^2(\frac{n_{tot}}{2S_{tot}^2}))$.*

Proof. We first rewrite $\frac{S_{1s}}{S_{tot}}$ to $\frac{n_{1s}R_{1s}}{n_{tot}R_{tot}}$. According to strong law of large numbers, $\frac{n_{1s}}{n_{tot}} = \frac{1}{2}$ as $n_{tot} \to \infty$ (see Fig. 3), and so $\frac{S_{1s}}{S_{tot}} = \frac{R_{1s}}{2R_{tot}}$. By Lemma 6, R_{1s} follows a normal distribution $\mathcal{N}(R_{tot}, \frac{\sigma_S^2}{n_{1s}})$, so the expectation of $\frac{S_{1s}}{S_{tot}}$ is: $\mu_{\frac{S_{1s}}{S_{tot}}} = \mathbb{E}[\frac{S_{1s}}{S_{tot}}] = \mathbb{E}[\frac{R_{1s}}{2R_{tot}}] = \frac{\mathbb{E}[R_{1s}]}{2R_{tot}} = 0.5$, and the variance is $Var[\frac{S_{1s}}{S_{tot}}] = Var[\frac{R_{1s}}{2\frac{S_{tot}}{n_{tot}}}] = Var[R_{1s}](\frac{n_{tot}}{2S_{tot}})^2 = \frac{\sigma_S^2}{n_{1s}}(\frac{n_{tot}^2}{4S_{tot}^2}) = \frac{\sigma_S^2}{0.5n_{tot}}(\frac{n_{tot}^2}{4S_{tot}^2}) = \sigma_S^2(\frac{n_{tot}}{2S_{tot}^2})$.

The corresponding standard deviation $\sigma_{\frac{S_{1s}}{S_{tot}}}$ is:

$$\sigma_{\frac{S_{1s}}{S_{tot}}} = \sqrt{Var[\frac{S_{1s}}{S_{tot}}]} = \frac{\sqrt{2}}{2}\sigma_S\frac{\sqrt{n_{tot}}}{S_{tot}} \quad □$$

Remark. *When $S_{tot} \to \infty$ in a given time interval, $\sigma_{\frac{S_{1s}}{S_{tot}}} \to 0$, and $\frac{S_{1s}}{S_{tot}} \to 0.5$. Moreover, we can use standard normal distribution table to analyze the desired false positive rate (FPR), that is, $FPR = 1-\Phi(w)$, where $\Phi(w)$ is the cumulative distribution function for the standard Normal distribution $\mathcal{N}(0,1)$. For example, if w is set to 2, the probability of $|\frac{S_{1s}}{S_{tot}} - \mu_{\frac{S_{1s}}{S_{tot}}}| \leq 2\sigma_{\frac{S_{1s}}{S_{tot}}}$ is 95%, and FPR should be only 5%. While $w = 3$ is chosen, FPR is just 0.3%.*

III. Secure and Robust Cardinality Estimation

In this section, we present CARBINE, an approach for secure and robust flow cardinality estimation in monitoring servers, which is built on top of HyperLogLog.

A. Threat models

We consider the case that HyperLogLog is in a black box, that is, the attacker does not know the hash function used in HyperLogLog. The hash functions are not typical hash functions, and they are usually accompanied with a specific seed, which is commonly used in modern hashes (e.g. xxhash [20]). Unlike regular hash function hashes flow key id with $h(id)$, changing the seed s of the hash functions $h(id, s)$ will completely change the output. Nevertheless, the attacker can insert fake flows into HyperLogLog to check the flow cardinality output (e.g. fuzz testing [21]). This can be easily done if the attacker knows that some specific applications using HyperLogLog are deployed in monitoring servers, such as Redis [14], Spark [16], and Presto [15]. The attacker only needs to install these applications on a server and test the estimated flow cardinality by crafting the flow key, such as varying the spoofed source IP and/or port. In this paper, we focus on two possible threat models:

M1 - Inflating attacks: The attacker adds a sequence of flows into HyperLogLog, of which the hashed flow id $h(id, s)$ can generate large values $v(x)$. The generated large $v(x)$ will significantly increment the values of some register cells, which causes large bias of flow cardinality estimation. This may lead to a lot of false positives for network monitoring servers to detect network anomalies.

M2 - Evasion attacks: The attacker has a set of flows $\mathcal{F}$ with cardinality n_{attack} and wants that they do not increment the cardinality estimation of HyperLogLog. This would be achieved if for all the flows id in $\mathcal{F}$, the hashed value $v(x) = 1$ or other small $v(x)$ below the values in the register cell M_j. Therefore, in this case, the flows in $\mathcal{F}$ would not increment any HyperLogLog register cells, and it is unlikely to affect the cardinality estimates at all for large cardinality. The malicious traffic generated by attackers can thereby evade the detection of HyperLogLog-based monitoring system.

B. Detection

In this section, we report how CARBINE detects possible threats. The workflow of CARBINE in the monitoring server is shown in Fig. 4: there are two m-sized HyperLogLog registers in CARBINE. Apart from the original HyperLogLog register M, there is a backup HyperLogLog register $\tilde{M}$. The unique difference of them is that they are using different hash functions, that is, *h()* for original HyperLogLog and *g()* for backup HyperLogLog. Algorithm 1 describes how the HyperLogLog register (i.e., M or $\tilde{M}$) is updated and queried. Each incoming packet has a flow key *id*, and we can retrieve a value $v(x)$ from the binary value of hashed id (Lines 7-8). If $v(x)$ is greater than the real-time minimum k_{min} but smaller than or equal to the base maximum k_{max}^{base}, then j is collected from hashed id, and $v(x)$ is compared to M_j, that is, the value at the index j of HyperLogLog register M (Lines 9-10). k_{min} and k_{max}^{base} are updated based on the sum of register values, i.e., $\sum_{i=0}^{m-1} M_j$. This can help us to prevent a large number of unnecessary accesses to the HyperLogLog register when $v(x) < k_{min}$ or $v(x) > k_{max}^{base}$. If $v(x)$ is larger than M_j, we add the incremented value (i.e. $v(x) - M_j$) to the sum and increment k_{min} and k_{max}^{base} if meeting the condition (Lines 11-16). Note that the HyperLogLog registers in CARBINE are still mergeable if the HyperLogLog registers in other monitoring servers use the same hash function and seed. In addition to the updates on HyperLogLog, any incoming packet increments the counter S_{tot}. If the packet has $v(x) = 1$, it also increments the counter S_{1s}.

M1 detection As proven in Theorem 3, with a probability at least $\frac{m-1}{m}$, $v(x)$ in HyperLogLog register should not be larger than k_{max}^{base}. Therefore, if $v(x)$ is greater than real-time base maximum k_{max}^{base} during time interval, this can be considered as an M1 threat.

M2 detection According to theorem 7, ideally, as the number of all incoming packets is large enough, the packets with $v(x) = 1$ should be the half. Therefore, we denote S_{1s} by $\widetilde{S}_{1s} = (0.5 + \theta)\widetilde{S}_{tot}$, where θ is a parameter to represent the fluctuations and $\mathbb{P}(|\theta| \leq 3\sigma_{\frac{S_{1s}}{S_{tot}}}) \geq 99.7\%$. However,

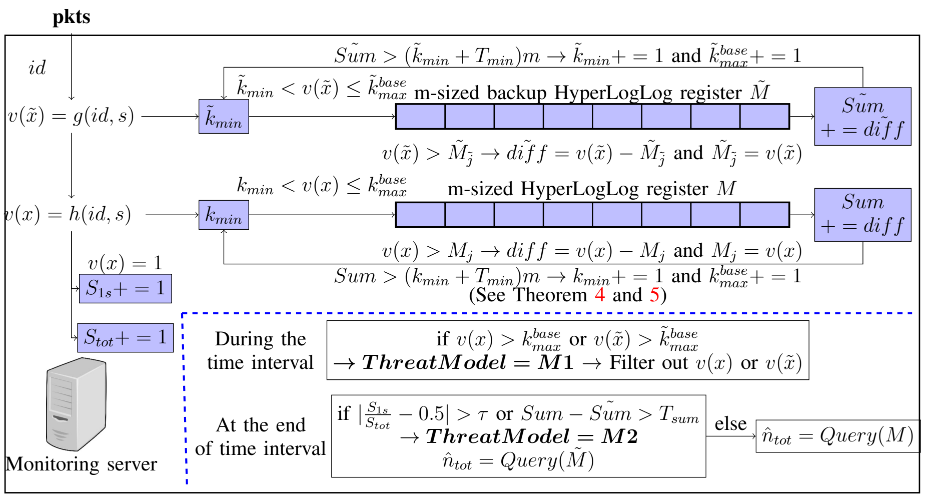

pkts
id
$\widetilde{Sum} > (\tilde{k}_{min} + T_{min})m \rightarrow \tilde{k}_{min} += 1$ and $\tilde{k}^{base}_{max} += 1$
$\tilde{k}_{min} < v(\tilde{x}) \leq \tilde{k}^{base}_{max}$ m-sized backup HyperLogLog register $\tilde{M}$
$v(\tilde{x}) = g(id, s)$
$\tilde{k}_{min}$
$\widetilde{Sum}$ $+= \widetilde{diff}$
$v(\tilde{x}) > \tilde{M}_{\tilde{j}} \rightarrow \widetilde{diff} = v(\tilde{x}) - \tilde{M}_{\tilde{j}}$ and $\tilde{M}_{\tilde{j}} = v(\tilde{x})$
$k_{min} < v(x) \leq k^{base}_{max}$ m-sized HyperLogLog register M
$v(x) = h(id, s)$
k_{min}
Sum $+= diff$
$v(x) > M_j \rightarrow diff = v(x) - M_j$ and $M_j = v(x)$
$Sum > (k_{min} + T_{min})m \rightarrow k_{min} += 1$ and $k^{base}_{max} += 1$
(See Theorem 4 and 5)
$v(x) = 1$
$S_{1s} += 1$
$S_{tot} += 1$
Monitoring server
During the time interval
if $v(x) > k^{base}_{max}$ or $v(\tilde{x}) > \tilde{k}^{base}_{max}$ $\rightarrow$ ThreatModel = M1 $\rightarrow$ Filter out $v(x)$ or $v(\tilde{x})$
At the end of time interval
if $|\frac{S_{1s}}{S_{tot}} - 0.5| > \tau$ or $Sum - \widetilde{Sum} > T_{sum}$ $\rightarrow$ ThreatModel = M2 $\hat{n}_{tot} = Query(\tilde{M})$
else
$\hat{n}_{tot} = Query(M)$

Fig. 4: Scheme of CARBINE in monitoring server

Algorithm 1: CARBINE

```
$m \leftarrow 2^l$ $(l \in \{4, ..., 16\})$, $Sum \leftarrow 0$, $s \leftarrow 1$
$M \leftarrow m$-sized HyperLogLog register
$T_{min} = \lfloor \log_2 \log_2 m \rfloor + 1.33$, $k_{min} \leftarrow 0$
$k^{base}_{max} \leftarrow k_{min} + \lfloor \log_2 \log_2 m \rfloor + \log_2 m - 1$
Function UpdateHLL($M$):
    for Each packet with flow key id do
        $t \leftarrow (Hash(id, s) \rightarrow \{0,1\}^{64})$
        $v(x) \leftarrow$ leftmost one of $t[l : l + 31]$+1
        if $k_{min} < v(x) \leq k^{base}_{max}$ then
            $j \leftarrow t[0 : l - 1]$
            if $v(x) > M_j$ then
                $diff \leftarrow v(x) - M_j$, $M_j \leftarrow v(x)$
                $Sum \leftarrow Sum + diff$
                if $Sum > (T_{min} + k_{min})m$ then
                    $k_{min} \leftarrow k_{min} + 1$
                    $k^{base}_{max} \leftarrow k^{base}_{max} + 1$
Function QueryHLL($M$):
    $\hat{n}_{tot} \leftarrow \alpha^{HLL}_m \cdot m^2 \cdot \sum_{j=0}^{m-1} 2^{-M_j}$
```

when an evasion attack happens, even though the values in the register do not increment, the packet count with small $v(x)$ still increases. This will break the regular packet count probability distribution of $v(x)$. Given such an observation, we propose a new method to detect M2 evasion attack.

Being $\widetilde{S}_{tot}$ the number of incoming packets in legitimate traffic and $\widetilde{S}_{1s}$ the number of packets with $v(x) = 1$ in a given time interval , assuming that the attack traffic volume is S_A, in which a fraction α $(0 \leq \alpha \leq 1)$ of S_A are with $v(x) = 1$, the ratio between the number of packets with $v(x) = 1$ counted in register S_{1s} and S_{tot} can be expressed as: $\frac{S_{1s}}{S_{tot}} = \frac{\widetilde{S}_{1s}+S_A}{\widetilde{S}_{tot}+S_A} = \frac{(0.5+\theta)\widetilde{S}_{tot}+\alpha S_A}{\widetilde{S}_{tot}+S_A} = 0.5+\theta+\frac{(\alpha-0.5-\theta)S_A}{\widetilde{S}_{tot}+S_A} = 0.5+\theta+\frac{\alpha-0.5-\theta}{\frac{\widetilde{S}_{tot}}{S_A}+1}$. Intuitively, $\varphi_A = |\frac{\alpha-0.5-\theta}{\frac{\widetilde{S}_{tot}}{S_A}+1}| > 0$ means that an evasion attack is taking place, and so an M2 alarm is triggered if:

$$|\frac{S_{1s}}{S_{tot}} - 0.5| > \tau = |\theta + \varphi_A|$$

By Theorem 7, if $|\theta+\varphi_A| > 2\sigma_{\frac{S_{1s}}{S_{tot}}}$, the attack can be detected with only 5% false positive rate. While $\sigma_{\frac{S_{1s}}{S_{tot}}}$ is computed from $\frac{\sqrt{2}}{2}\sigma_S\frac{\sqrt{n_{tot}}}{S_{tot}}$, the network operator can analyze the previous long-term flow statistics in the network to approximate the variance in a given time interval. This would not be a limitation for our case since the traffic matrix in ISP networks is usually stable, and we will also show this fact in our experimental evaluation. Supposing $|\theta| \leq 2\sigma_{\frac{S_{1s}}{S_{tot}}}$, $|\varphi_A|$ should be greater than $4\sigma_{\frac{S_{1s}}{S_{tot}}}$ so that $|\theta+\varphi_A| > 2\sigma_{\frac{S_{1s}}{S_{tot}}}$ is always true. It can be also noted that a higher attack traffic volume S_A leads to a larger φ_A, and the attack is easier to get detected. The appropriate network scenario for our M2 detection can be a volumetric (i.e. high-packet-rate) DDoS attack that evades flow cardinality control in the monitoring server using HyperLogLog.

We assume that the DDoS attacker generates a bunch of packets with $v(x) = 1$ to different register cells. This is the most common case for attackers since attacker does not know the minimum of HyperLogLog register. Such a simple attack has been discussed in [22] and demonstrated to be effective. Thus, in this case, $\alpha = 1$, and the attack is detected if $|\varphi_A| = |\frac{0.5-\theta}{\frac{\widetilde{S}_{tot}}{S_A}+1}| \geq 4\sigma_{\frac{S_{1s}}{S_{tot}}}$. For instance, while $\sigma_{\frac{S_{1s}}{S_{tot}}} = 0.015$, $\frac{S_A}{\widetilde{S}_{tot}} \geq 0.14$, which means that the volumetric DDoS attack with attack traffic volume S_A greater than 14% of total volume $\widetilde{S}_{tot}$ can always be detected.

We would not ignore the possible limitation of our approach: if the fraction α of packets with $v(x) = 1$ is closer to 0.5 (e.g. the half of attack traffic volume in the time interval is with $v(x) = 1$ and the other half is with $v(x) = 2$), the attack will be almost impossible to get detected. However, we believe that this case only rarely happens in black box case: since the attacker does not know any information (e.g. minimum and $v(x)$) inside HyperLogLog, the most reliable way for attackers is to generate the flow keys that do not increment the flow cardinality estimations in any testing HyperLogLog instances. This means that the resulted flow keys are mostly with $v(x) = 1$. Moreover, as proven in Theorem 6, if the difference of the sum of two HyperLogLog registers in CARBINE is greater than a given threshold T_{sum} (i.e. w times of $\sqrt{7.02m}$), we also trigger an M2 threat alarm.

C. Protection

M1 protection Any inflating attack will be pro-actively filtered out by our CARBINE since the values $v(x)$ greater than k^{base}_{max} are not considered in the HyperLogLog register. In this way, CARBINE is robust to extremely large values. In our evaluation (see Section IV-C), we will also show that removing $v(x) > k^{base}_{max}$ can help improve the accuracy of HyperLogLog.

M2 protection If an alarm of M2 is triggered, at the end of time interval, collector coordinates all monitoring servers in the network to query the backup HyperLogLog register $\tilde{M}$ to perform network-wide flow cardinality estimation. This means that it is possible to recover the actual flow cardinality estimation even though the HyperLogLog register is under evasion attack. Meanwhile, collector needs to force monitoring servers to change the seed in the hash function used for HyperLogLog for the following time intervals.

IV. Evaluation

We implemented our entire CARBINE strategy in Python to study its performance over simulated network scenario.

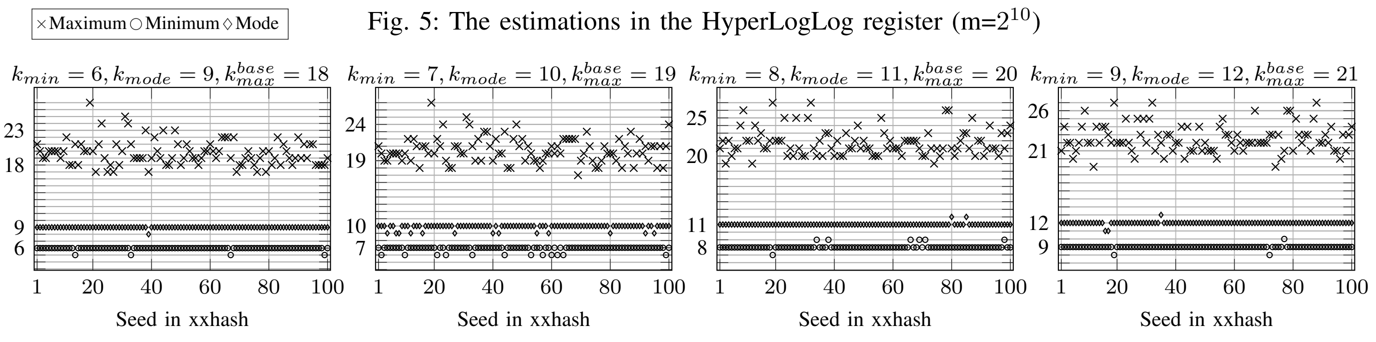


Fig. 5: The estimations in the HyperLogLog register (m=2^{10})

(a) $T_{int} = 5s$ (b) $T_{int} = 10s$ (c) $T_{int} = 30s$ (d) $T_{int} = 60s$

A. Testing flow traces and default settings

In our simulations, we used three 10-minutes CAIDA passive flow traces collected from a 10 Gbps backbone link, in which CAIDA 2016 [23] contains 300 million packets, CAIDA 2018 [24] contains 276 million packets, and CAIDA 2019 [25] contains 360 million packets. We first split the CAIDA 2019 trace into different time intervals (i.e. 5, 10, 30 and 60 seconds) to test proposed solutions on different number of packets. If not otherwise specified, the default flow key is 5 tuple (i.e. source and destination IP, source and destination port, and protocol), and the length of time interval is 5 seconds. We then ran for 120 measurement intervals to investigate the M2 detection performance in consecutive time intervals considering all three CAIDA flow traces. The hash functions used for HyperLogLog is xxhash [20]. The 32-bit seed s can be varied to generate completely different outputs, and by default it is set to 1. The default HyperLogLog register size m is $2^{10} = 1024$, and the register cell size is 5 bits each. As a result, the threshold T_{sum} is $2\cdot\sqrt{7.02 \cdot 1024} = 84$.

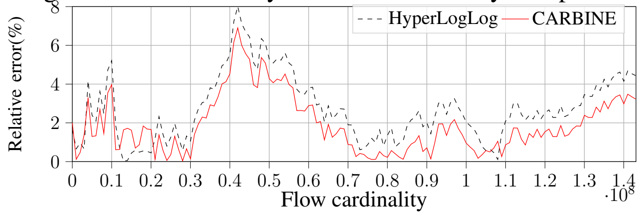
B. Evaluation of maximum, minimum, and mode estimations

Fig. 5 shows the maximum, minimum and mode estimations of HyperLogLog register values varying seed in xxhash function in different time intervals. The seed in the hash function is varied from 1 to 100. Clearly, larger time intervals contain higher number of distinct flows. In all four figures, the mode k_{mode} and the minimum k_{min} are mostly inline with our theoretical analysis: in the worst case (i.e. $T_{int} = 10s$), the estimation accuracy is still greater than 87%, and the estimation bias of k_{mode} and k_{min} is at most 1. As proven in Theorem 1, the bias does not completely depend on the flow cardinality n_{tot} but $\frac{n_{tot}}{m2^k}$: when $e^{-\frac{n_{tot}}{m2^k}}$ and $e^{-\frac{n_{tot}}{m2^{k-1}}}$ are close to 0.5, both k and $k-1$ have a high probability to be the mode k_{mode}. This is why $T_{int} = 10s$ has larger number of flows than $T_{int} = 5s$ but the estimation accuracy is lower. The resulted maximum value k_{max} of HyperLogLog register depends on the hash function: as we explained in Theorem 3, there is a non-negligible probability (i.e. $\frac{1}{m}$) that the maximum value of HyperLogLog register is $k_{max}^{base} + u$ ($u \in \mathbb{N}^{+}$). Unlike mode and minimum, if a flow key is hashed into a large value, this will replace the maximum value of HyperLogLog within a whole time interval. This is why most resulted maximum value is greater than k_{max}^{base}.

C. Evaluation of flow cardinality estimation accuracy

In this subsection, we would like to show that filtering out large values $v(x)$ (i.e. $v(x) > k_{max}^{base}$) not only improves

Fig. 6: Flow cardinality estimation accuracy comparison



the robustness of CARBINE to inflating attack (M1 threat), but also performs even higher accuracy on flow cardinality estimation. As the described query function of HyperLogLog is to estimate flow cardinality between $\frac{5}{2}m$ (i.e. 2560) and $\frac{1}{30}2^{32} \approx 1.43\cdot10^8$, in our evaluation the tested flow cardinality range is [12400, 143012400], and the step length is 10000. Therefore, in total there are 14300 points shown in Fig. 6. The results show that CARBINE has lower relative error than HyperLogLog in most cases when varying flow cardinality. However, in scenarios where HyperLogLog contains a small number of extremely large values and the majority of values are close to the mode, HyperLogLog may achieve slightly better accuracy than CARBINE. Numerically, the average relative error of CARBINE in those points is 1.94%, whereas that of HyperLogLog is 2.76%. This reveals that CARBINE does not degrade the flow cardinality estimation performance of HyperLogLog, but even better.

D. Case study: volumetric DDoS attacks evading increments in HyperLogLog

In this subsection, we examine a network facing volumetric DDoS attacks that try to evade the detection of monitoring system using HyperLogLog, and network operators can utilize our approach to detect M2 threats. We analyzed the detection performance in terms of attack traffic volume and then compared our approach to a state-of-the-art work considering several real volumetric DDoS attack flow traces.

1) Evaluation metrics: We consider *true-positive rate* D_{TP}, *false-positive rate* D_{FP} and *detection accuracy* D_{acc} as evaluation metrics of DDoS detction. Given that *(i)* True Positive (TP) is the number of time intervals detecting M2 threat while a DDoS attack is occurring in 120 time intervals, *(ii)* True Negative (TN) is the number of time intervals without triggering any M2 threat detection while no DDoS attack is occurring, *(iii)* False Positive (FP) is the number of time intervals detecting M2 threat while no DDoS attack is occurring, and *(iv)* False Negative (FN) is the number of time intervals without triggering M2 threat detection while a DDoS

Fig. 7: The value of $\frac{S_{1s}}{S_{tot}} \sim \mathcal{N}(\mu{=}0.5, \sigma^2{=}0.015^2)$ varying synthetic attack traffic volume (CAIDA2019)

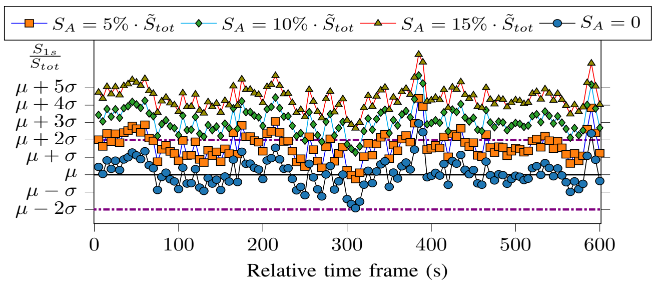


TABLE I: Volumetric DDoS detection performance varying attack traffic volume (flow key = 5 tuple)

Flow trace name	**Attack volume** S_A	**False positive** D_{FP}	**True-positive rate** D_{TP}	**Detection accuracy** D_{acc}
CAIDA2019 ($\tau = 0.030$)	$5\% \cdot \widetilde{S}_{tot}$	2.5% (3/120)	30.83% (37/120)	64.16%
	$7.5\% \cdot \widetilde{S}_{tot}$		70.83% (85/120)	84.17%
	$10\% \cdot \widetilde{S}_{tot}$		95% (114/120)	96.25%
	$12.5\% \cdot \widetilde{S}_{tot}$		99.17% (119/120)	98.33%
	$15\% \cdot \widetilde{S}_{tot}$		100% (120/120)	98.75%
CAIDA2016 ($\tau = 0.028$)	$5\% \cdot \widetilde{S}_{tot}$	0% (0/120)	35% (42/120)	67.50%
	$7.5\% \cdot \widetilde{S}_{tot}$		81.67% (98/120)	90.83%
	$10\% \cdot \widetilde{S}_{tot}$		98.33% (118/120)	99.17%
	$12.5\% \cdot \widetilde{S}_{tot}$		100% (120/120)	100%
	$15\% \cdot \widetilde{S}_{tot}$		100% (120/120)	100%
CAIDA2018 ($\tau = 0.036$)	$5\% \cdot \widetilde{S}_{tot}$	0% (0/120)	14.17% (17/120)	57.08%
	$7.5\% \cdot \widetilde{S}_{tot}$		53.33% (64/120)	76.67%
	$10\% \cdot \widetilde{S}_{tot}$		92.50% (111/120)	96.25%
	$12.5\% \cdot \widetilde{S}_{tot}$		99.17% (119/120)	99.58%
	$15\% \cdot \widetilde{S}_{tot}$		100% (120/120)	100%

attack is instead occurring, the metrics introduced above are defined as: $D_{TP} = \frac{TP}{TP+FN} \times 100\%$, $D_{FP} = \frac{FP}{TN+FP} \times 100\%$, and $D_{acc} = \frac{TP+TN}{TP+TN+FP+FN} \times 100\%$.

2) DDoS detection performance varying attack traffic volume: We synthesized several flow traces with the attack traffic volume S_A varying from 5% to 15% of the total traffic volume $\widetilde{S}_{tot}$ in 120 5-seconds CAIDA 2019 flow trace, and the $v(x)$ of flows in the attack traffic S_A are always hashed to 1. As a result, the fraction between S_{1s} and S_{tot} can be formulated as $\frac{S_{1s}}{S_{tot}} = \frac{\widetilde{S}_{1s}+S_A}{\widetilde{S}_{tot}+S_A}$. The threshold τ is set to $2\bar{\sigma}_{\frac{S_{1s}}{S_{tot}}}$, where $\bar{\sigma}_{\frac{S_{1s}}{S_{tot}}}$ is the median value of $\sigma_{\frac{S_{1s}}{S_{tot}}}$ in 120 time intervals, that is, 0.015. This means that if $|\frac{S_{1s}}{S_{tot}} - 0.5| > 2\bar{\sigma}_{\frac{S_{1s}}{S_{tot}}} = 0.03$, it is considered as an M2 threat. Fig. 7 shows that when there is no attack traffic (i.e. $S_A = 0$), in only 3 among 120 measurements, $\frac{S_{1s}}{S_{tot}}$ is out of $\mu + 2\sigma$, which illustrates that the false positive rate in this experiment is 2.5%. Intuitively, $\frac{S_{1s}}{S_{tot}}$ increases as the traffic volume increases, and so the true positive rate. While the attack traffic volume reaches 12.5% of total volume $\widetilde{S}_{tot}$, the true positive rate is 99.17% and the corresponding detection accuracy is 98.33%. This is clearly shown in Table.I. The same experiments are also carried out in flow trace CAIDA 2016 and CAIDA 2018 with the same experimental settings. In both CAIDA 2016 and CAIDA 2018, CARBINE behaves similar detection performance as the experiments in CAIDA 2019, but in CAIDA 2016 CARBINE can achieve even higher detection accuracy than the other two cases since the flow distribution in CAIDA 2016 is more stable, i.e., the median of its standard deviation $\bar{\sigma}_{\frac{S_{1s}}{S_{tot}}}$ is the smallest among all flow traces.

3) Detection performance comparing to a state-of-the-art approach considering real DDoS flow traces: In this sub-

TABLE II: Volumetric DDoS detection performance comparison (flow key = 5 tuple)

Flow trace name	**DDoS trace name**	**No. packets** S_A (% of $\widetilde{S}_{tot}$)	**No. flows** n_A (% of $\widetilde{n}_{tot}$)	D_{TP} **of CARBINE**	D_{TP} **of SHLL** [18]
CAIDA2019 ($\tau = 0.030$)	Booter1	$\sim$ 400000 ($\sim$ 13%)	$\sim$ 22000 ($\sim$ 7%)	100%	50%
	Booter4	$\sim$ 320000 ($\sim$ 10%)	$\sim$ 117000 ($\sim$ 35%)	100%	100%
	Booter6	$\sim$ 450000 ($\sim$ 15%)	$\sim$ 145000 ($\sim$ 43%)	100%	100%
	Booter7	$\sim$ 170000 ($\sim$ 5%)	$\sim$ 77000 ($\sim$ 23%)	100%	100%

TABLE III: Volumetric DDoS Detection performance comparison (flow key = $\{srcIP, dstIP\}$)

Flow trace name	**DDoS trace name**	**No. packets** S_A (% of $\widetilde{S}_{tot}$)	**No. flows** n_A (% of $\widetilde{n}_{tot}$)	D_{TP} **of CARBINE**	D_{TP} **of SHLL** [18]
CAIDA2019 ($\tau = 0.0306$)	Booter 1	$\sim$ 400000 ($\sim$ 13%)	$\sim$ 3000 ($\sim$ 1.15%)	98.33%	3.33%
	Booter 4	$\sim$ 320000 ($\sim$ 10%)	$\sim$ 2800 ($\sim$ 1.08%)	98.33%	2.50%
	Booter 6	$\sim$ 450000 ($\sim$ 15%)	$\sim$ 7000 ($\sim$ 2.69%)	100%	9.17%
	Booter 7	$\sim$ 170000 ($\sim$ 5%)	$\sim$ 5800 ($\sim$ 2.23%)	68.33%	8.33%
CAIDA2016 ($\tau = 0.0285$)	Booter 1	$\sim$ 400000 ($\sim$ 16%)	$\sim$ 3000 ($\sim$ 2.00%)	100%	14.16%
	Booter 4	$\sim$ 320000 ($\sim$ 13%)	$\sim$ 2800 ($\sim$ 1.86%)	100%	14.16%
	Booter 6	$\sim$ 450000 ($\sim$ 18%)	$\sim$ 7000 ($\sim$ 4.66%)	100%	36.67%
	Booter 7	$\sim$ 170000 ($\sim$ 7%)	$\sim$ 5800 ($\sim$ 3.86%)	96.67%	30.83%
CAIDA2018 ($\tau = 0.0380$)	Booter 1	$\sim$ 400000 ($\sim$ 17%)	$\sim$ 3000 ($\sim$ 1.57%)	95.83%	6.67%
	Booter 4	$\sim$ 320000 ($\sim$ 14%)	$\sim$ 2800 ($\sim$ 1.47%)	86.67%	6.67%
	Booter 6	$\sim$ 450000 ($\sim$ 19%)	$\sim$ 7000 ($\sim$ 3.68%)	98.33%	10.83%
	Booter 7	$\sim$ 170000 ($\sim$ 8%)	$\sim$ 5800 ($\sim$ 3.05%)	49.16%	9.16%

section, we consider DDoS attack traffic in the real world, namely *Booter*. Booter services [26] are on-demand services that provide support for illegal users to launch DDoS attacks targeting websites and network servers. We picked four UDP-based DNS amplification attack flow traces (i.e. Booter 1, 4, 6, 7 collected from a 10Gbps link) that can better reflect the difference between our CARBINE and the existing solution [18]. The attack traffic volume in each flow trace is greater than 100Mbps, which is the most common metric to determine a volumetric DDoS attack [27]. The existing solution, for short we name it *SHLL*, used two HyperLogLog registers with different hash functions to check if the relative error of them is greater than a threshold. The relative error proved by SHLL approximately follows a normal distribution $\mathcal{N}(0, \frac{1}{m_1} + \frac{1}{m_2})$, where m_1 and m_2 are the size of two HyperLogLog registers. Hence, similar to our approach, the threshold can also be chosen as w times of the standard deviation $\sqrt{\frac{1}{m_1} + \frac{1}{m_2}}$. For a fair comparison, both HyperLogLog register size m_1 and m_2 in SHLL are set to 2^{10}, and w is 2. Therefore, the detection threshold in SHLL is $2 \cdot \sqrt{2^{-10} + 2^{-10}} = 0.09$. We captured 120 5-seconds long of packets from DDoS attack flow traces according to their timestamps, and inserted the packets into our testing CAIDA flow traces depending on the time interval index. The properties of captured DDoS traces are reported in Table.II and Table.III. Since the theoretical false positive rate of both approaches is 5%, we only focus on comparing the true positive rate D_{TP}. The same as last subsection, all packets in the attack traffic are hashed with $v(x) = 1$. The first HyperLogLog register in both CARBINE and SHLL uses $xxhash$ with seed 1, while the second of them uses the same hash function with seed 2. Initially, as shown in Table.II, we

TABLE IV: Average update speed in 50 times of tests

Performance	Algorithm	# Distinct items ($\times$ 1024)			
		10^2	10^3	10^4	10^5
Update speed (Mups)	HyperLogLog	13.68	13.43	13.13	12.87
	CARBINE	13.85	13.62	13.36	13.05

used 5 tuple as the flow key, the true positive rate D_{tp} of both CARBINE and SHLL keeps 100% when CAIDA 2019 trace under the attack of Booter 4, 6, and 7 (attack traffic greater than 20% of legitimate traffic). For Booter 1, the D_{tp} of CARBINE remains 100%, but that of SHLL is only 50%. This is because the distinct flows in attack traffic of Booter 1 is only approximately 7% of legitimate traffic, which is the lowest among all attack traffic. This tells us that SHLL has good performance on detecting evasion attacks only when the number of distinct flows in the attack traffic is large. The attacks on CAIDA 2016 and 2018 behave the same as those in CAIDA 2019, and we omitted the detailed results of the other two flow traces for conciseness.

However, as shown in Table.III, when the considered flow keys in HyperLogLog are {srcIP, dstIP} pairs, SHLL cannot effectively detect potential attacks any more. This table indicates that the performance of SHLL depends on not only the number of flows in attack traffic but also the flow cardinality in legitimate traffic. The true positive rate of SHLL becomes even worse since the number of flows in the normal traffic is larger, and the percentage of attack flows is relatively smaller. The performance of CARBINE becomes slightly worse when we switch the flow key from 5 tuple to {srcIP, dstIP} due to the small increase of τ, but it still completely outperforms SHLL especially when the attack traffic packets percentage is greater than 10%, CARBINE outperforms SHLL. In CAIDA 2016, even though the proportion of attack traffic volume is only 7%, due to the low variance of this flow trace, both CARBINE and SHLL can perform relatively high true positive rate than that of the other two flow traces.

This demonstrates that, for M2 threats, CARBINE has good detection performance on DDoS attacks with large attack traffic volume ($\geq$ 10% of legitimate traffic) or large number of distinct flows, while SHLL is only able to detect DDoS attacks if the attack packets are categorized with fine-grained flow keys (e.g. 5 tuple).

E. Evaluation of update speed

We additionally implemented HyperLogLog and CARBINE in C language and measured their speed in a server with two-core Intel(R) Xeon(R) Gold 6234 CPU 3.30GHz and 376GB RAM. We used two HyperLogLogs in CARBINE, but the results of a HyperLogLog do not rely on the other. Therefore, they can be updated in parallel using multiple CPU logical cores, and clearly this will not be the speed bottleneck of CARBINE if enough CPU resource is assigned. As reported in Table IV, the average million updates per second (Mups) in 50-times tests of both algorithms decreases as the number of distinct items increases. This is because more distinct items lead to more updates in the register when $v(x) > M_j$. Since *(i.)* comparing $v(x)$ to the real-time minimum k_{min} and base maximum k_{max}^{base} can filter out a large number of unnecessary accesses to HyperLogLog and *(ii.)* no register scan is required to retrieve k_{min} and k_{max}^{base}, CARBINE performs even higher update speed than HyperLogLog when varying the number of distinct items, i.e., approximately 0.2 million more items can be processed in a second.

V. Related work

Sketch-based flow cardinality estimation for monitoring Many sketch-based algorithms for estimating the cardinality of data streams have been proposed in literature, including Linear Counting [28], Multiresolution Bitmap [29], PCSA [30], LogLog [19] and HyperLogLog [13]. Estimating the cardinality of large network data streams using sketches directly in the server has become an appealing solution to enhance network monitoring due to its low memory occupation and high accuracy. It has been proven that HyperLogLog is able to achieve the same accuracy as other methods but requires much less memory, which has been widely used in many query engines for big network data, including Redis [14], Spark [16], and Presto [15]. Intuitively, using aforementioned Sketches is efficient to monitor the number of active flows on a high speed link. In addition, many recent works [31] [5] [32] [12] use them to identify network anomalies: a sudden large increase of distinct flows (i.e. connections) targeting a specific destination host may indicate that a DDoS attack is taking place. Similarly, the superspreaders [33] [34] (e.g. worm propagation [35]) can be identified if the same source host is contacting a significant number of destination hosts.

Security and robustness of HyperLogLog While existing works focus on the improvement of the accuracy [36] or memory efficiency [37] of HyperLogLog, how to protect HyperLogLog is becoming a new challenge. Reviriego *et al.* [18] recently investigated the evasion attack (M2 threat) of HyperLogLog and proposed a method to detect such a kind of attack by comparing two estimations queried from two HyperLogLog registers using different hash functions. If the relative error of two estimations is larger than a given threshold, then the evasion attack is considered as taking place. However, this comes with a limitation: their approach is effective only when the flow number of attack traffic is large. Paterson *et al.* [22] comprehensively analyzed the potential security issues of HyperLogLog, but they did not provide any sufficient solutions to defend against them. Our CARBINE overcomes the constraints of HyperLogLog from an algorithmic perspective: CARBINE not only can effectively detect attacks evading increments in HyperLogLog, but also can pro-actively filter out inflating attacks without sacrificing any estimation accuracy or update speed of HyperLogLog.

VI. Conclusion

In this paper, we started with the theoretical analysis of HyperLogLog and explored its additional properties. We then studied two possible threats of HyperLogLog, and proposed corresponding solutions leveraging explored new properties. Considering proposed solutions as building blocks, we present CARBINE, a novel approach that improves the security and robustness of HyperLogLog while increasing the estimation accuracy and update speed. The findings indicate that CARBINE can proficiently identify potential threats, such as volumetric DDoS attacks using evasion techniques, thereby safeguarding HyperLogLog.

Acknowledgement

The author would like to thank Domenico Siracusa, Marco Savi, Federico Pederzolli, and Qingjun Xiao for the early discussion and their insightful comments. The author also wants to thank Noa Zilberman for her encouragement leading to the final submission of this paper. This research was partly funded by EU Horizon SmartEdge (Grant agreement No. 101092908, Innovate UK 10056403). For the purpose of Open Access, the author has applied a CC BY public copyright license to any Author Accepted Manuscript (AAM) version arising from this submission.

References

[1] R. B. Basat, G. Einziger, S. L. Feibish, J. Moraney, and D. Raz, "Network-wide routing-oblivious heavy hitters," in *IEEE/ACM Symposium on Architectures for Networking and Communications Systems (ANCS)*. New York, NY, USA: ACM, 2018.

[2] R. Harrison, Q. Cai, A. Gupta, and J. Rexford, "Network-wide heavy hitter detection with commodity switches," in *ACM Symposium on SDN Research (SOSR)*, 2018.

[3] D. Ding, M. Savi, G. Antichi, and D. Siracusa, "An incrementally-deployable p4-enabled architecture for network-wide heavy-hitter detection," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 75–88, 2020.

[4] D. Ding, M. Savi, and D. Siracusa, "Estimating logarithmic and exponential functions to track network traffic entropy in p4," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–9.

[5] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2016.

[6] Â. C. Lapolli, J. A. Marques, and L. P. Gaspary, "Offloading real-time ddos attack detection to programmable data planes," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2019, pp. 19–27.

[7] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2018.

[8] D. Ding, M. Savi, and D. Siracusa, "Tracking normalized network traffic entropy to detect ddos attacks in p4," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 4019–4031, 2021.

[9] X. Jing, Z. Yan, and W. Pedrycz, "Security data collection and data analytics in the internet: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 586–618, 2018.

[10] Y. Zhou, Y. Zhou, M. Chen, and S. Chen, "Persistent spread measurement for big network data based on register intersection," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 1, pp. 1–29, 2017.

[11] H. Huang, Y.-E. Sun, S. Chen, S. Tang, K. Han, J. Yuan, and W. Yang, "You can drop but you can't hide: k-persistent spread estimation in high-speed networks," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1889–1897.

[12] D. Ding, M. Savi, F. Pederzolli, M. Campanella, and D. Siracusa, "In-network volumetric ddos victim identification using programmable commodity switches," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1191–1202, 2021.

[13] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," in *Discrete Mathematics and Theoretical Computer Science*, pp. 137-156, 2007.

[14] Redis, "https://redis.io/."

[15] Presto, "https://prestosql.io/."

[16] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi *et al.*, "Spark sql: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. New York, NY, USA: ACM, 2015, pp. 1383–1394.

[17] Y. Chabchoub, R. Chiky, and B. Dogan, "How can sliding hyperloglog and ewma detect port scan attacks in ip traffic?" *EURASIP Journal on Information Security*, vol. 2014, no. 1, p. 5, 2014.

[18] P. Reviriego and D. Ting, "Security of hyperloglog (hll) cardinality estimation: Vulnerabilities and protection," *IEEE Communications Letters*, vol. 24, no. 5, pp. 976–980, 2020.

[19] M. Durand and P. Flajolet, "Loglog counting of large cardinalities," in *European Symposium on Algorithms*. Springer, 2003, pp. 605–617.

[20] Y. Collet, "xxhash–extremely fast hash algorithm," https://github.com/Cyan4973/xxHash.

[21] G. Klees, A. Ruef, B. Cooper, S. Wei, and M. Hicks, "Evaluating fuzz testing," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2123–2138.

[22] K. G. Paterson and M. Raynal, "Hyperloglog: Exponentially bad in adversarial settings," in *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2022, pp. 154–170.

[23] CAIDA UCSD Anonymized Internet Traces Dataset -[passive-2016], "http://www.caida.org/data/passive/passive_dataset.xml," 2016.

[24] CAIDA UCSD Anonymized Internet Traces Dataset -[passive-2018], "http://www.caida.org/data/passive/passive_dataset.xml," 2018.

[25] CAIDA UCSD Anonymized Internet Traces Dataset -[passive-2019], "http://www.caida.org/data/passive/passive_dataset.xml," 2019.

[26] J. Santanna, R. van Rijswijk-Deij, R. Hofstede, A. Sperotto, M. Wierbosch, L. Zambenedetti Granville, and A. Pras, "Booters - An analysis of DDoS-as-a-service attacks," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015.

[27] Volumetric DDoS attacks, "https://www.radware.com/security/ddos-knowledge-center/ddos-chronicles/ddos-attacks-history/," 2020.

[28] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Transactions on Database Systems (TODS)*, vol. 15, no. 2, pp. 208–229, 1990.

[29] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high speed links," in *ACM SIGCOMM conference on Internet measurement*, 2003.

[30] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *Journal of computer and system sciences*, vol. 31, no. 2, pp. 182–209, 1985.

[31] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.

[32] Q. Huang, X. Jin, P. P. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang, "Sketchvisor: Robust network measurement for software packet processing," in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2017.

[33] L. Tang, Q. Huang, and P. P. Lee, "Spreadsketch: Toward invertible and network-wide detection of superspreaders," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1608–1617.

[34] X. Jing, Z. Yan, H. Han, and W. Pedrycz, "Extendedsketch: Fusing network traffic for super host identification with a memory efficient sketch," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 3913–3924, 2021.

[35] E. Levy, "Worm propagation and generic attacks," *IEEE Security & Privacy*, vol. 3, no. 2, pp. 63–65, 2005.

[36] S. Heule, M. Nunkesser, and A. Hall, "Hyperloglog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm," in *Proceedings of the 16th International Conference on Extending Database Technology*, 2013, pp. 683–692.

[37] Q. Xiao, Y. Zhou, and S. Chen, "Better with fewer bits: Improving the performance of cardinality estimation of large data streams," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.