

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

Damu Ding

University of Oxford, UK & ByteDance Inc., US

21st May, 2024



What is flow cardinality?

- Counting distinct items

What is flow cardinality?

- Counting distinct items

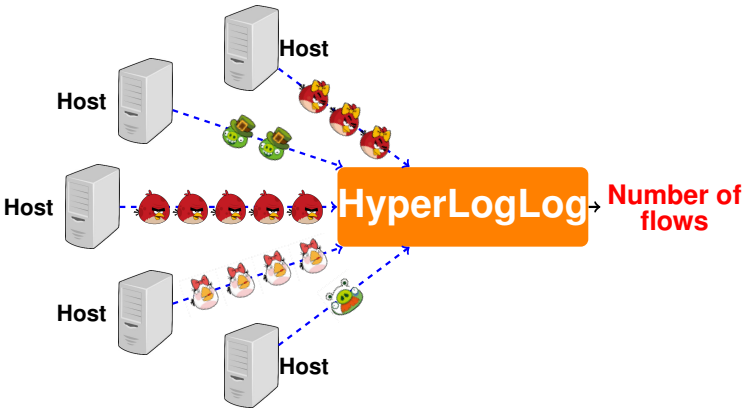
- Number of distinct items: **3**

What is flow cardinality?

- Counting distinct items

- Number of distinct items: **3**
- **Flow cardinality**: the number of distinct flows identified by different flow keys (e.g. src IP, dst IP)

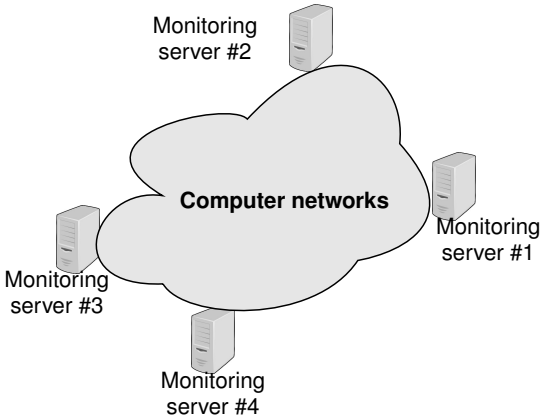
Estimate overall number of flows



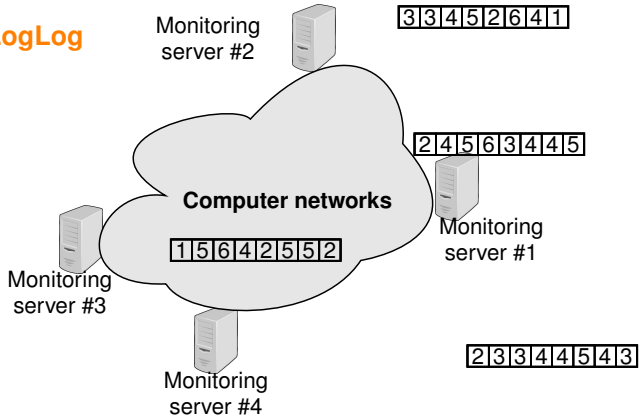
- **Fast:** Time complexity is only O(1)
- **Efficient and accurate:** 2560 bytes can estimate 10^9 numbers with standard error below 2%.
- **Mergeable:** Multiple HyperLogLogs can be merged into one

Flajolet, Philippe, et al. "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm." Discrete mathematics & theoretical computer science Proceedings (2007).

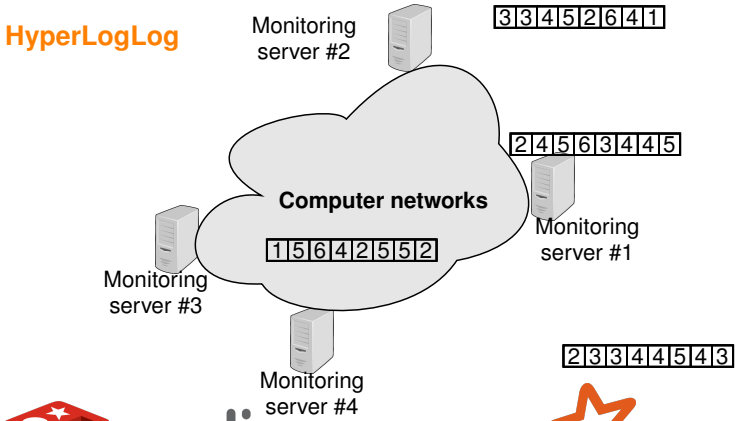
Flow cardinality estimation in network monitoring



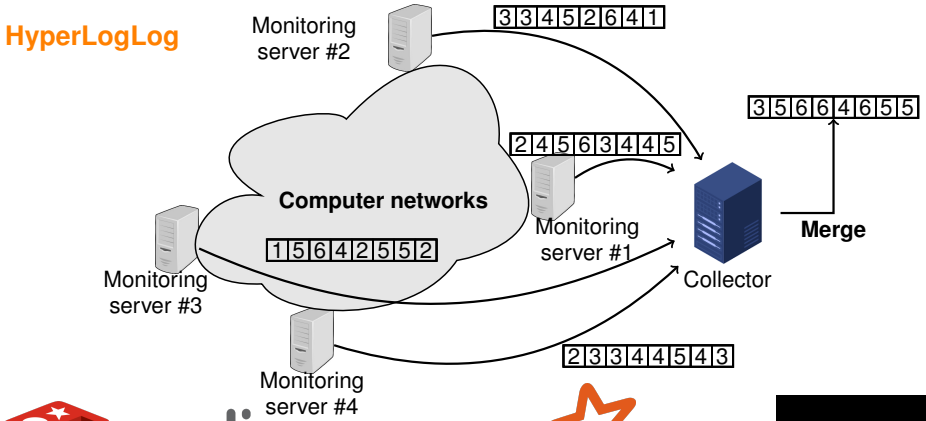
Flow cardinality estimation in network monitoring



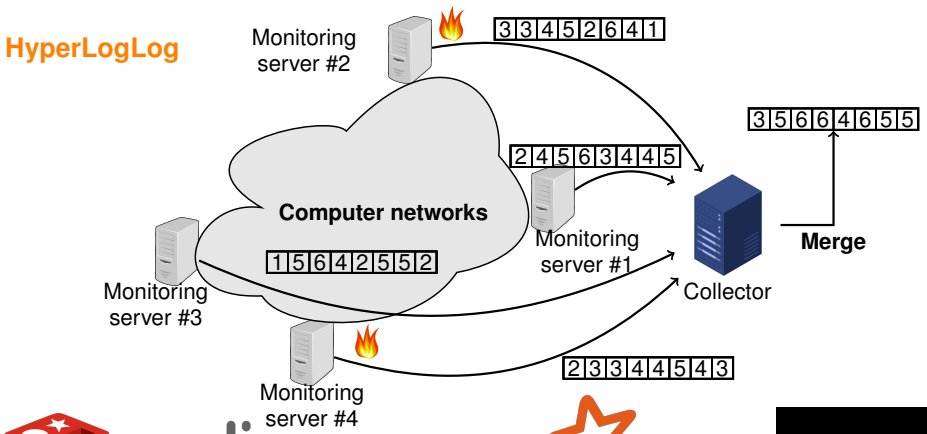
Flow cardinality estimation in network monitoring



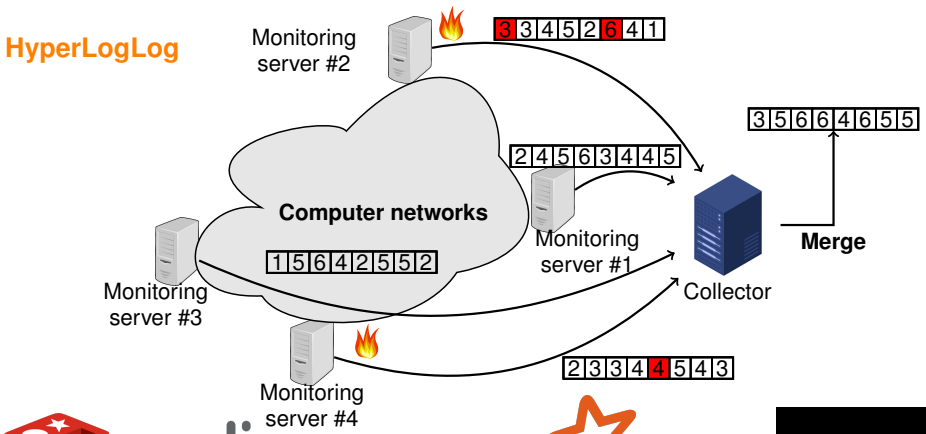
Flow cardinality estimation in network monitoring



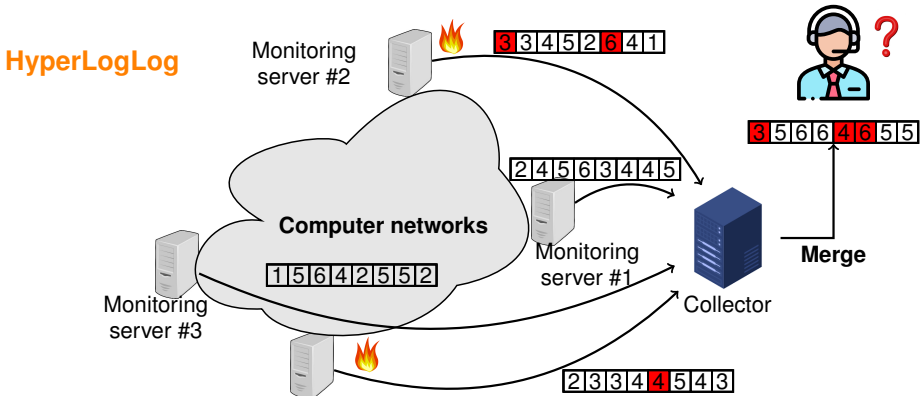
Flow cardinality estimation in network monitoring



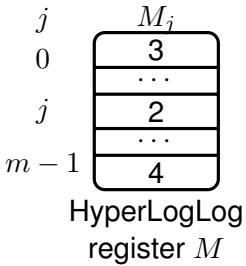
Flow cardinality estimation in network monitoring

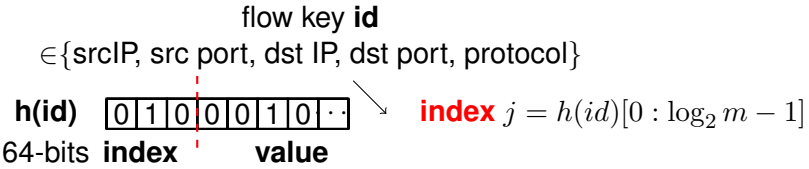


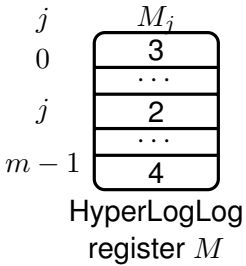
Flow cardinality estimation in network monitoring

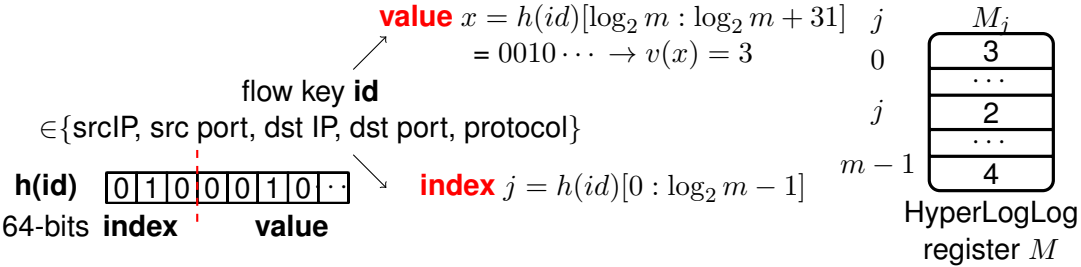


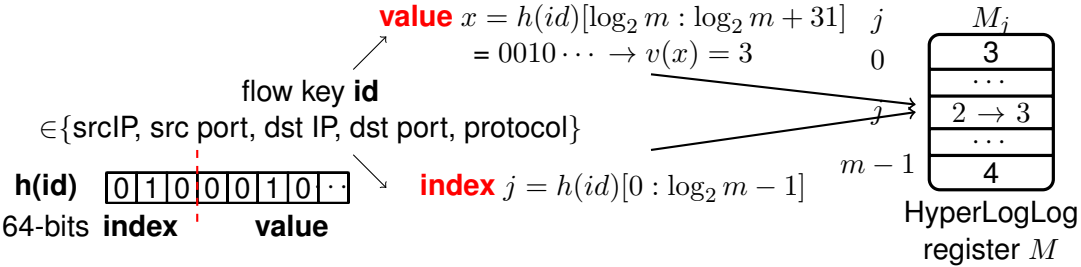
flow key **id**
$\in$ {srcIP, src port, dst IP, dst port, protocol}

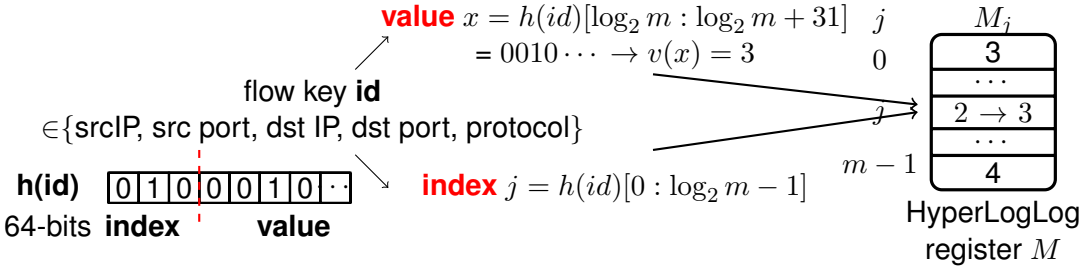



flow key **id**
∈{srcIP, src port, dst IP, dst port, protocol}
h(id)
64-bits **index** **value**
0 1 0 0 0 1 0 ··
index $j = h(id)[0 : \log_2 m - 1]$


j
M_j
0
3
···
j
2
···
$m-1$
4
HyperLogLog
register M


value $x = h(id)[\log_2 m : \log_2 m + 31]$
$= 0010\cdots \rightarrow v(x) = 3$
flow key id
$\in${srcIP, src port, dst IP, dst port, protocol}
h(id)
64-bits
index
value
0 1 0 0 0 1 0 ··
index $j = h(id)[0 : \log_2 m - 1]$
j
0
j
$m-1$
M_j
3
···
2
···
4
HyperLogLog
register M


value $x = h(id)[\log_2 m : \log_2 m + 31]$
$= 0010\cdots \rightarrow v(x) = 3$
flow key id
$\in${srcIP, src port, dst IP, dst port, protocol}
h(id)
64-bits
index
value
0 1 0 0 0 1 0 · ·
index $j = h(id)[0 : \log_2 m - 1]$
j
0
j
$m-1$
M_j
3
· · ·
$2 \rightarrow 3$
· · ·
4
HyperLogLog
register M



Query: $\hat{n} = \alpha_m^{HLL} m^2 (\sum_{j=0}^{m-1} (2^{-M(j)}))^{-1}$

- M: HyperLogLog register
- m: HyperLogLog register size
- α_m^{HLL}: Bias correction parameter depending on m

Threat models

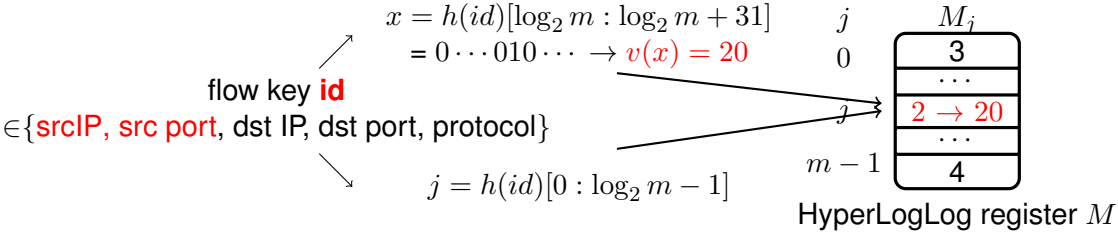
Assumptions:

- HyperLogLog is in the blackbox
  - The attacker does not know the hash function used in HyperLogLog
- Hash functions $h(id, s)$ in HyperLogLog are not typical hash functions $h(id)$
  - Changing the seed s of the hash functions $h(id, s)$ will completely change the output.
- The attacker can create Hyperloglog that use the same hash functions as the Hyperloglog in the target server being attacked
  - This can be easily done if the attacker knows some specific applications using HyperLogLog are deployed in monitoring servers, such as Redis, Spark, and Presto.

Threat models:

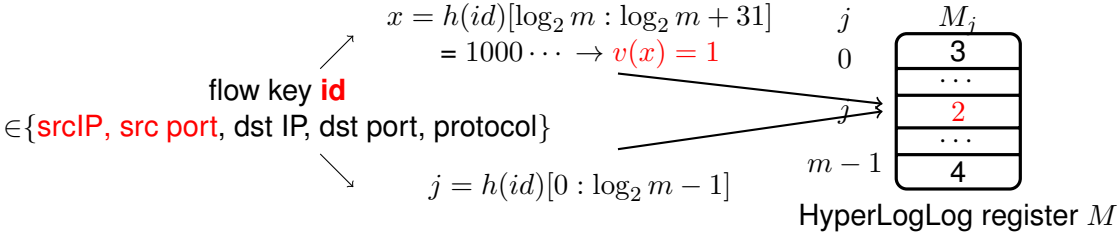
1. **M1**- Inflating attack (security)
2. **M2**- Evasion attack (security)

M1-Inflating attack



- The attacker inserts few flows to HyperLogLog, of which the hashed flow id $h(id)$ can generate high tailing-zeros values $v(x)$.

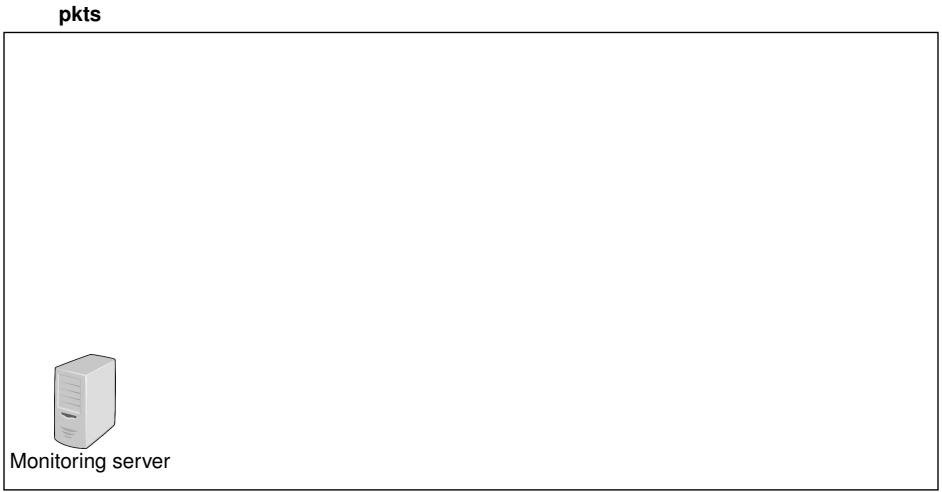
M2-Evasion attack



- The attacker can craft a set of flows $\mathcal{F}$ with $v(x) = 1$ or other small values and wants that they do not increment the cardinality estimation of HyperLogLog.

Scheme of CARBINE

Update

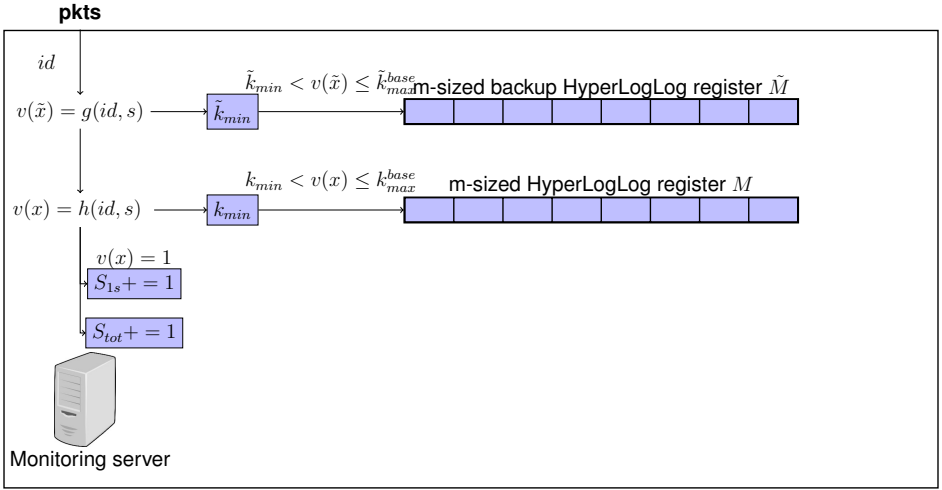


Scheme of CARBINE

Update

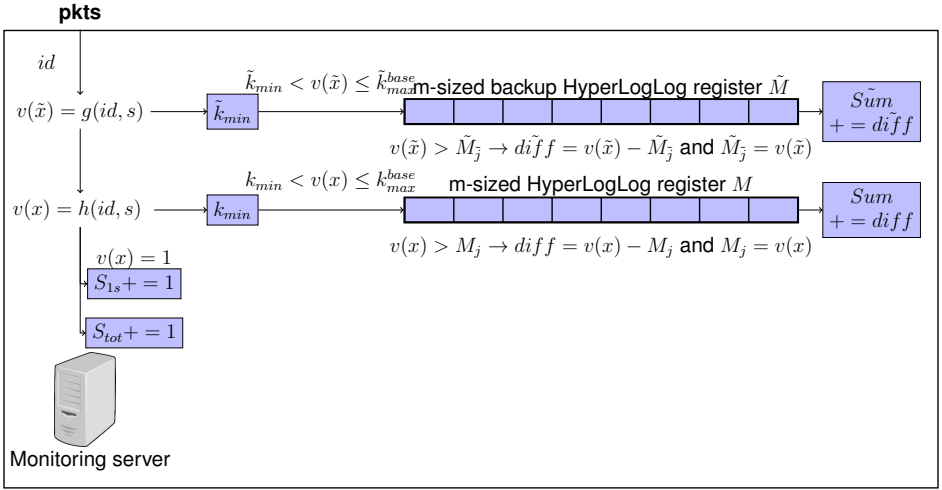
Scheme of CARBINE

Update



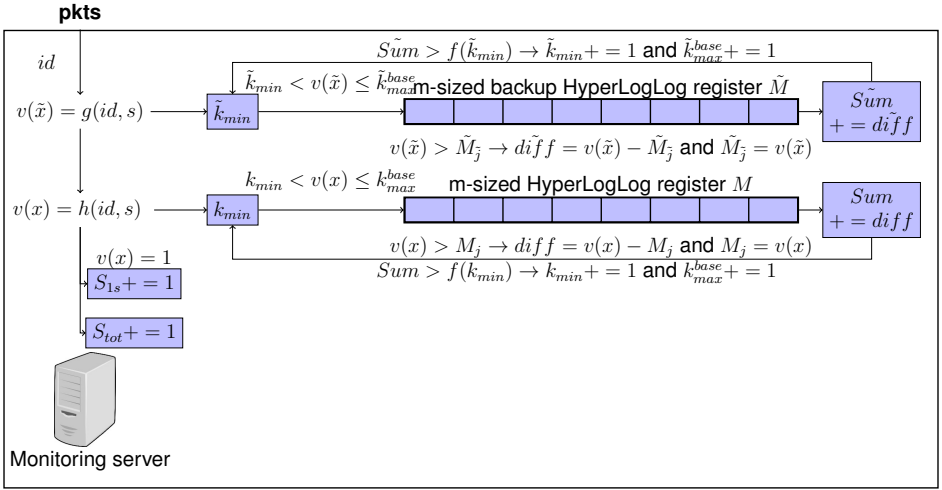
Scheme of CARBINE

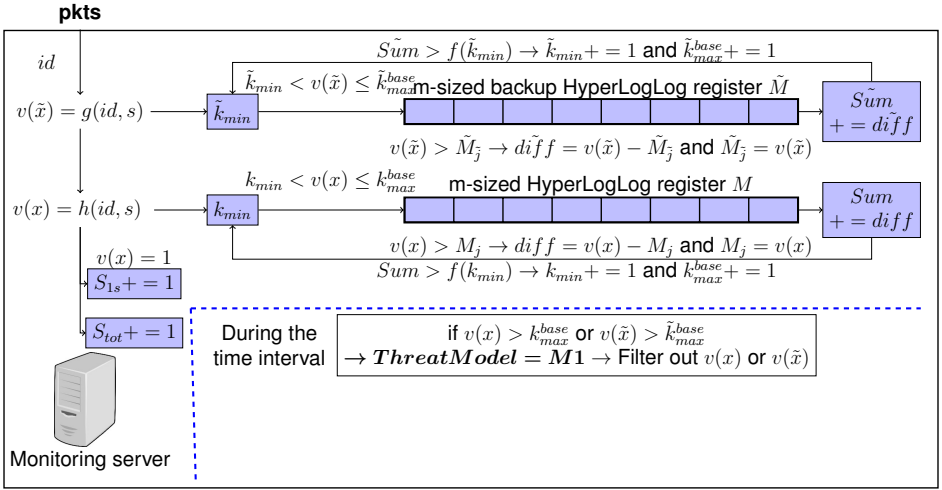
Update

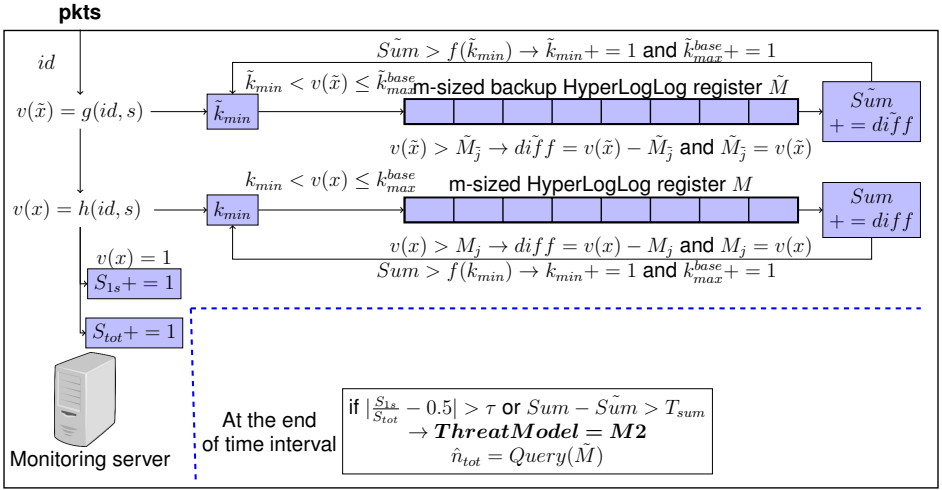


Scheme of CARBINE

Update

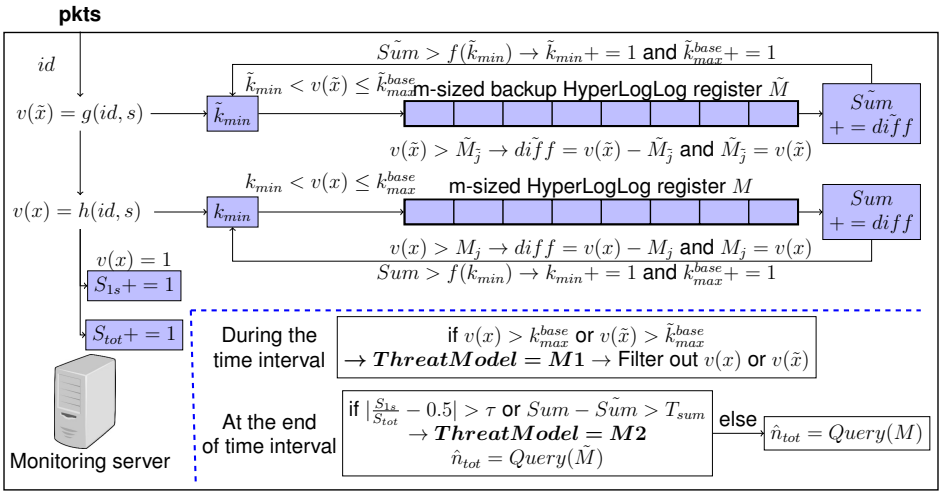



pkts
id
$v(\tilde{x}) = g(id, s)$
$v(x) = h(id, s)$
$v(x) = 1$
$S_{1s} += 1$
$S_{tot} += 1$
Monitoring server
$\tilde{k}_{min}$
k_{min}
$S\tilde{u}m > f(\tilde{k}_{min}) \rightarrow \tilde{k}_{min} += 1$ and $\tilde{k}^{base}_{max} += 1$
$\tilde{k}_{min} < v(\tilde{x}) \leq \tilde{k}^{base}_{max}$
m-sized backup HyperLogLog register $\tilde{M}$
$S\tilde{u}m += di\tilde{f}f$
$v(\tilde{x}) > \tilde{M}_j \rightarrow di\tilde{f}f = v(\tilde{x}) - \tilde{M}_j$ and $\tilde{M}_j = v(\tilde{x})$
$k_{min} < v(x) \leq k^{base}_{max}$
m-sized HyperLogLog register M
$Sum += diff$
$v(x) > M_j \rightarrow diff = v(x) - M_j$ and $M_j = v(x)$
$Sum > f(k_{min}) \rightarrow k_{min} += 1$ and $k^{base}_{max} += 1$
During the time interval
if $v(x) > k^{base}_{max}$ or $v(\tilde{x}) > \tilde{k}^{base}_{max}$
$\rightarrow \boldsymbol{ThreatModel = M1} \rightarrow$ Filter out $v(x)$ or $v(\tilde{x})$


pkts
id
$v(\tilde{x}) = g(id, s)$
$\tilde{k}_{min}$
$\tilde{k}_{min} < v(\tilde{x}) \leq \tilde{k}_{max}^{base}$
m-sized backup HyperLogLog register $\tilde{M}$
$S\tilde{u}m > f(\tilde{k}_{min}) \rightarrow \tilde{k}_{min} += 1$ and $\tilde{k}_{max}^{base} += 1$
$S\tilde{u}m += di\tilde{f}f$
$v(\tilde{x}) > \tilde{M}_j \rightarrow di\tilde{f}f = v(\tilde{x}) - \tilde{M}_j$ and $\tilde{M}_j = v(\tilde{x})$
$v(x) = h(id, s)$
k_{min}
$k_{min} < v(x) \leq k_{max}^{base}$
m-sized HyperLogLog register M
$Sum += diff$
$v(x) > M_j \rightarrow diff = v(x) - M_j$ and $M_j = v(x)$
$Sum > f(k_{min}) \rightarrow k_{min} += 1$ and $k_{max}^{base} += 1$
$v(x) = 1$
$S_{1s} += 1$
$S_{tot} += 1$
Monitoring server
At the end of time interval
if $|\frac{S_{1s}}{S_{tot}} - 0.5| > \tau$ or $Sum - S\tilde{u}m > T_{sum}$
$\rightarrow \boldsymbol{ThreatModel = M2}$
$\hat{n}_{tot} = Query(\tilde{M})$

Scheme of CARBINE

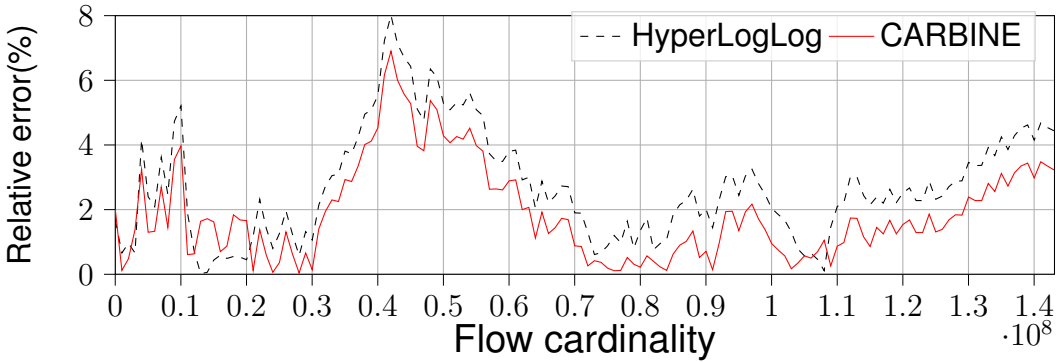
Query



HyperLogLog protection

1. M1 protection
   - Any inflating attack will be **pro-actively filtered out** by our CARBINE since the values $v(x)$ greater than k_{max}^{base} are not considered.
2. M2 protection
   - Query the **backup** HyperLogLog register $\tilde{M}$ to perform flow cardinality estimation.
   - Collector needs to force monitoring servers to **change the seed** in the hash function.
   - The flow cardinality estimation is recoverable even under evasion attack.
   - The **network-wide flow cardinality estimation** is still valid.

Flow cardinality estimation accuracy



- **Range:** [12400, 143012400] **Step length:** 10000 **Points:** 14300
- **Average relative error:** *CARBINE* 1.94% VS. *HyperLogLog* 2.76%
- CARBINE performs **even better** flow cardinality estimation performance when filtering out large values.

Case study: volumetric DDoS attacks

The metric of DDoS detection using HyperLogLog: exceed a threshold of distinct flows to a destination host or a group of destination hosts

- The attacker can evade increments in HyperLogLog to prevent DDoS detection

Case study: volumetric DDoS attacks

Evaluation metrics:

- True Positive (TP): the number of time intervals detecting M2 threat while a DDoS attack is occurring in 120 time intervals (10 minutes flow trace split into 5 seconds each)
- True Negative (TN): the number of time intervals without triggering any M2 threat detection while no DDoS attack is occurring
- False Positive (FP): the number of time intervals detecting M2 threat while no DDoS attack is occurring
- False Negative (FN): the number of time intervals without triggering M2 threat detection while a DDoS attack is instead occurring

$$D_{TP} = \frac{TP}{TP + FN} \times 100\% \quad D_{FP} = \frac{FP}{TN + FP} \times 100\%$$

$$D_{acc} = \frac{TP + TN}{TP + TN + FP + FN} \times 100\%$$

Case study: volumetric DDoS attacks

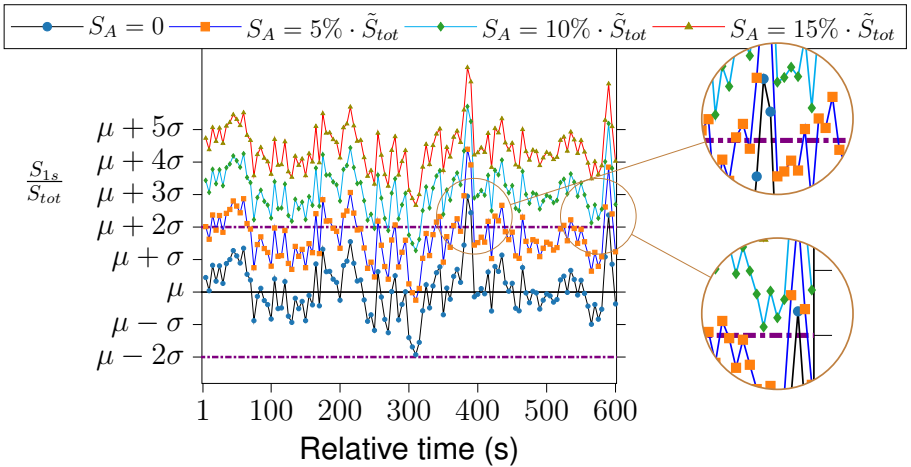


Figure: The value of $\frac{S_{1s}}{S_{tot}} \sim \mathcal{N}(\mu{=}0.5, \sigma^2{=}0.015^2)$ varying attack traffic volume (CAIDA2019)

Case study: volumetric DDoS attacks

Table: Volumetric DDoS detection performance varying attack volume

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

$\tau = 2\bar{\sigma}_{\frac{S_{1s}}{S_{tot}}}$, where $\bar{\sigma}_{\frac{S_{1s}}{S_{tot}}}$ is the median of $\sigma_{\frac{S_{1s}}{S_{tot}}}$ in 120 time intervals.

Case study: volumetric DDoS attacks

Table: Properties of DDoS flow traces[1] and detection performance comparing to state of the art [2] (flow key = $\{srcIP, dstIP\}$)

Flow trace name	DDoS trace name	No. packets (% of $\widetilde{S}_{tot}$)	No. flows (% of $\widetilde{n}_{tot}$)	D_{TP} of CARBINE	D_{TP} of SHLL[4]
CAIDA2019	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	$\sim$ 200000 ($\sim$ 6%)	$\sim$ 5800 ($\sim$ 2.23%)	68.33%	8.33%
CAIDA2016	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	$\sim$ 200000 ($\sim$ 8%)	$\sim$ 5800 ($\sim$ 3.86%)	96.67%	30.83%
CAIDA2018	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	$\sim$ 200000 ($\sim$ 9%)	$\sim$ 5800 ($\sim$ 3.05%)	49.16%	9.16%

[1] Santanna, José Jair, et al. "Booters—An analysis of DDoS-as-a-service attacks." 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM). IEEE, 2015.

[2] Reviriego, Pedro, and Daniel Ting. "Security of HyperLogLog (HLL) Cardinality Estimation: Vulnerabilities and Protection." IEEE Communications Letters 24.5 (2020): 976-980.

Update speed

Table: Average update speed in 50 times of tests (m=2^{10})

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

- 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
- **No register scan** is required to retrieve k_{min} and k_{max}^{base}
- CARBINE performs **even higher update speed** than original HyperLogLog

Conclusion

- Explore additional properties of HyperLogLog
  - The real-time maximum and minimum can be retrieved from the sum of HyperLogLog
- Investigate two threat models in HyperLogLog
  - M1: Inflating attack
  - M2: Evasion attack
- We present CARBINE to detect and protect HyperLogLog
  - More secure and robust
  - Better estimation accuracy
  - Higher update speed
- We evaluate our CARBINE by considering volumetric DDoS attacks within a practical network scenario.

Thank you!

damu.ding@bytedance.com