# Estimating Logarithmic and Exponential Functions to Track Network Traffic Entropy in P4


Damu Ding[1,2], Marco Savi[1,3], Domenico Siracusa[1]

[1] *Fondazione Bruno Kessler, Trento, Italy* [2] *University of Bologna, Bologna, Italy*
[3] *University of Milano-Bicocca, Milan, Italy*



**NOMS 2020**
21st April, 2020

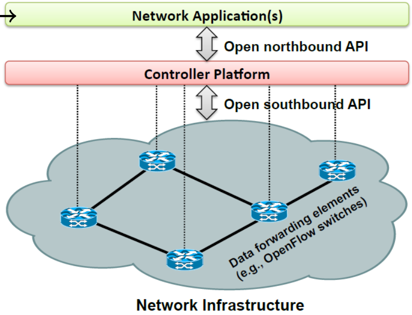
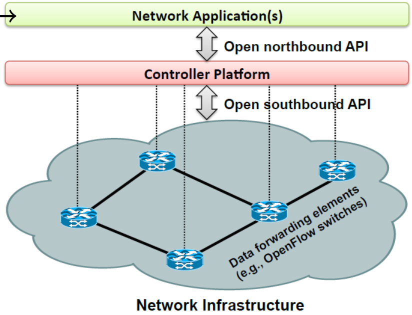



Figure source: Kreutz, Diego, et al. "Software-defined networking: A comprehensive survey." Proceedings of the IEEE 103.1 (2015): 14-76. and https://n0where.net/real-time-network-monitoring-cyberprobe

# Network monitoring/security functionalities in programmable data planes

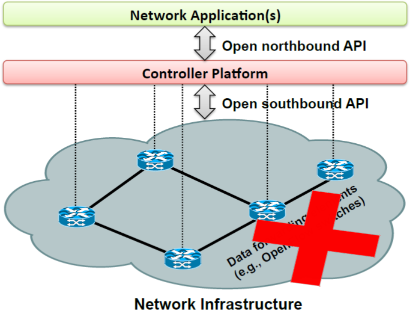



1. **Significant communication overhead**
2. **The latency caused by interaction**

---

Figure source: Kreutz, Diego, et al. "Software-defined networking: A comprehensive survey." Proceedings of the IEEE 103.1 (2015): 14-76. and https://n0where.net/real-time-network-monitoring-cyberprobe

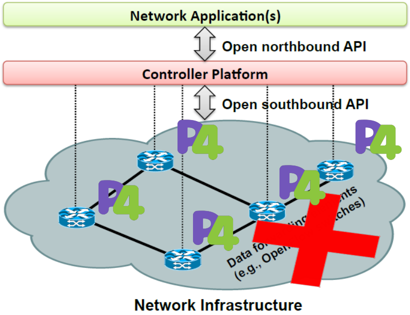
# Network monitoring/security functionalities in programmable data planes




**Data plane programmable switches**

Figure source: Kreutz, Diego, et al. "Software-defined networking: A comprehensive survey." Proceedings of the IEEE 103.1 (2015): 14-76. and https://n0where.net/real-time-network-monitoring-cyberprobe

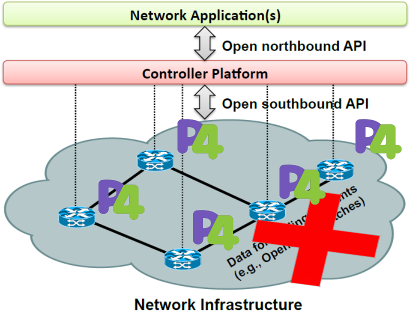
# Network monitoring/security functionalities in programmable data planes




Data plane programmable switches

Figure source: Kreutz, Diego, et al. "Software-defined networking: A comprehensive survey." Proceedings of the IEEE 103.1 (2015): 14-76. and https://n0where.net/real-time-network-monitoring-cyberprobe

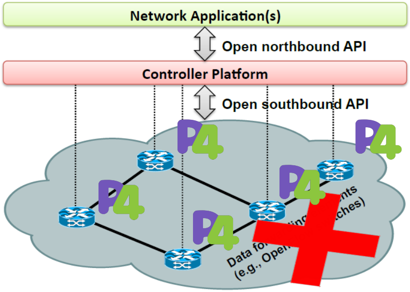
# Network monitoring/security functionalities in programmable data planes




**Data plane programmable switches**

Figure source: Kreutz, Diego, et al. "Software-defined networking: A comprehensive survey." Proceedings of the IEEE 103.1 (2015): 14-76. and https://n0where.net/real-time-network-monitoring-cyberprobe

# Network monitoring/security functionalities in programmable data planes




**Heavy-hitter detection**

**Flow cardinality estimation**

**DDoS detection**

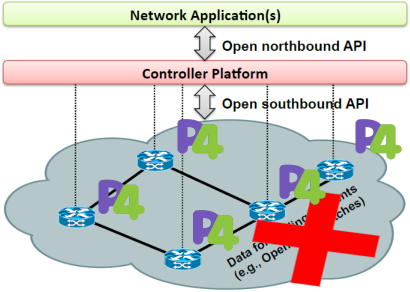
**Network traffic entropy estimation**

**Data plane programmable switches**

---

Figure source: Kreutz, Diego, et al. "Software-defined networking: A comprehensive survey." Proceedings of the IEEE 103.1 (2015): 14-76. and https://n0where.net/real-time-network-monitoring-cyberprobe

# Network monitoring/security functionalities in programmable data planes




**Heavy-hitter detection**

**Flow cardinality estimation**

**DDoS detection**

**Network traffic entropy estimation**

**Data plane programmable switches**

---

Figure source: Kreutz, Diego, et al. "Software-defined networking: A comprehensive survey." Proceedings of the IEEE 103.1 (2015): 14-76. and https://n0where.net/real-time-network-monitoring-cyberprobe

# Network traffic entropy

- **Shannon entropy** $H = -\sum_{i=1}^{n} \frac{f_i}{|S|_{tot}} \log_2 \frac{f_i}{|S|_{tot}}$
  - $f_i$: the packet count of the incoming flow $i$
  - $|S|_{tot}$: the total number of processed packets by the switch during time interval
  - $n$: the overall number of flows

# Network traffic entropy

- **Shannon entropy** $H = -\sum_{i=1}^{n} \frac{f_i}{|S|_{tot}} \log_2 \frac{f_i}{|S|_{tot}}$
  - $f_i$: the packet count of the incoming flow $i$
  - $|S|_{tot}$: the total number of processed packets by the switch during time interval
  - $n$: the overall number of flows
- Network traffic entropy $H$ gives an indication on **traffic distribution** across the network
  - Minimum $H = 0$: when all packets $|S|_{tot}$ belong to the same flow $i$
  - Maximum $H = \log_2 n$: when $n$ flows are uniform distributed

# Network traffic entropy

- **Shannon entropy** $H = -\sum_{i=1}^{n} \frac{f_i}{|S|_{tot}} \log_2 \frac{f_i}{|S|_{tot}}$
  - $f_i$: the packet count of the incoming flow $i$
  - $|S|_{tot}$: the total number of processed packets by the switch during time interval
  - $n$: the overall number of flows
- Network traffic entropy $H$ gives an indication on **traffic distribution** across the network
  - Minimum $H = 0$: when all packets $|S|_{tot}$ belong to the same flow $i$
  - Maximum $H = \log_2 n$: when $n$ flows are uniform distributed
- A periodical track of entropy helps diagnose security and performance issues
  - DDoS detection
    - The entropy of dstIP significantly decreases
  - Port-scan detection
    - The entropy of {dst IP, dst port} significantly decreases

# Network traffic entropy

- **Shannon entropy** $H = -\sum_{i=1}^{n} \frac{f_i}{|S|_{tot}} \log_2 \frac{f_i}{|S|_{tot}}$
  - $f_i$: the packet count of the incoming flow $i$
  - $|S|_{tot}$: the total number of processed packets by the switch during time interval
  - $n$: the overall number of flows
- Network traffic entropy $H$ gives an indication on **traffic distribution** across the network
  - Minimum $H = 0$: when all packets $|S|_{tot}$ belong to the same flow $i$
  - Maximum $H = \log_2 n$: when $n$ flows are uniform distributed
- A periodical track of entropy helps diagnose security and performance issues
  - DDoS detection
    - The entropy of dstIP significantly decreases
  - Port-scan detection
    - The entropy of $\{$dst IP, dst port$\}$ significantly decreases

**Is it implementable in P4-enabled programmable switches?**

# P4 language limitations

- P4 is a high-level domain-specific language for programmable switches.
- Supports $+$, $-$, $*$, $\gg$, $\ll$, $\hat{}$ (XOR), $|$ (OR), & (AND), if-else statements, etc.

# P4 language limitations

- P4 is a high-level domain-specific language for programmable switches.
- Supports $+$, $-$, $*$, $\gg$, $\ll$, $\hat{}$ (XOR), $|$ (OR), & (AND), if-else statements, etc.

Logarithmic function

# P4 language limitations

- P4 is a high-level domain-specific language for programmable switches.
- Supports $+$, $-$, $*$, $\gg$, $\ll$, $\hat{}$ (XOR), $|$ (OR), & (AND), if-else statements, etc.

$\boxed{\text{Exponential function}}$ $\boxed{\text{Logarithmic function}}$

# P4 language limitations

- P4 is a high-level domain-specific language for programmable switches.
- Supports $+$, $-$, $*$, $\gg$, $\ll$, $\hat{}$ (XOR), $|$ (OR), & (AND), if-else statements, etc.

$\boxed{\text{Loops (}\textit{For/While}\text{ )}}$

$\boxed{\text{Exponential function}}$ $\boxed{\text{Logarithmic function}}$

# P4 language limitations

- P4 is a high-level domain-specific language for programmable switches.
- Supports $+$, $-$, $*$, $\gg$, $\ll$, ^ (XOR), | (OR), & (AND), if-else statements, etc.

Division

Loops (*For*/*While* )

Exponential function

Logarithmic function

# P4 language limitations

- P4 is a high-level domain-specific language for programmable switches.
- Supports $+$, $-$, $*$, $\gg$, $\ll$, ^ (XOR), | (OR), & (AND), if-else statements, etc.

Floating numbers

Division

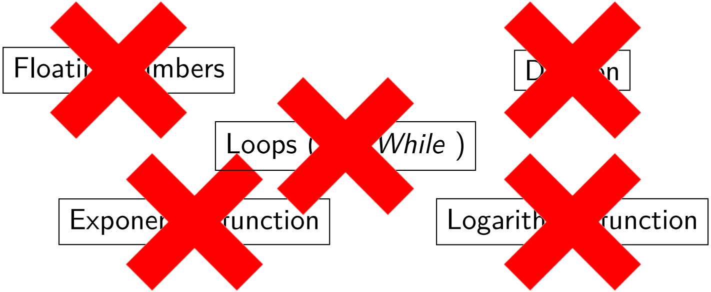
Loops (*For*/*While* )

Exponential function

Logarithmic function

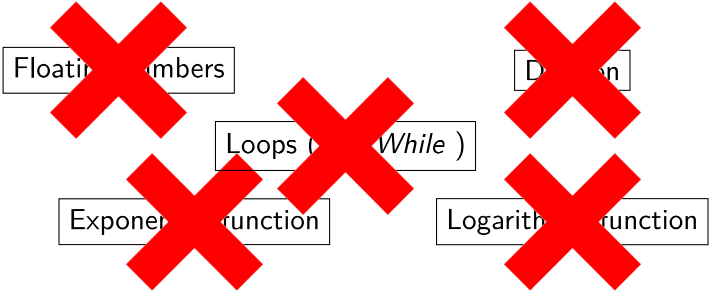
# P4 language limitations

- P4 is a high-level domain-specific language for programmable switches.
- Supports $+$, $-$, $*$, $\gg$, $\ll$, ^ (XOR), | (OR), & (AND), if-else statements, etc.



# P4 language limitations

- P4 is a high-level domain-specific language for programmable switches.
- Supports $+$, $-$, $*$, $\gg$, $\ll$, $\hat{}$ (XOR), $|$ (OR), & (AND), if-else statements, etc.




- Enable logarithmic and exponential-function estimation as the building blocks for network traffic entropy estimation entirely in P4

# Why logarithmic and exponential-function estimations?

# Why logarithmic and exponential-function estimations?

- Network traffic entropy estimation

$$H = -\sum_{i=1}^{n} \frac{f_i}{|S|_{tot}} \log_2 \frac{f_i}{|S|_{tot}} = \log_2 |S|_{tot} - \frac{1}{|S|_{tot}} \sum_{i=1}^{n} f_i \log_2 f_i$$
$$= \log_2(|S|_{tot}) - 2^{(\sum_{i=1}^{n} f_i \log_2 f_i - \log_2 |S|_{tot})}$$

**3** times of logarithmic computation and **1** time of exponential-function computation

# Why logarithmic and exponential-function estimations?

- Network traffic entropy estimation

$$H = -\sum_{i=1}^{n} \frac{f_i}{|S|_{tot}} \log_2 \frac{f_i}{|S|_{tot}} = \log_2 |S|_{tot} - \frac{1}{|S|_{tot}} \sum_{i=1}^{n} f_i \log_2 f_i$$
$$= \log_2(|S|_{tot}) - 2^{(\sum_{i=1}^{n} f_i \log_2 f_i - \log_2 |S|_{tot})}$$

**3** times of logarithmic computation and **1** time of exponential-function computation

- SOTA [1] needs thousands of table entries in TCAM for the logarithmic and exponential-function estimations to assure the relative error is under 1%
  - TCAM is expensive and power-hungry
  - SOTA needs a controller to populate the TCAM lookup tables in the switch

[1] Sharma, N. K., Kaufmann et al. "Evaluating the power of flexible packet processing for network resource allocation" Symposium on Networked Systems Design and Implementation (NSDI 17) (pp. 67-82).

# Why logarithmic and exponential-function estimations?

- Network traffic entropy estimation

$$H = -\sum_{i=1}^{n} \frac{f_i}{|S|_{tot}} \log_2 \frac{f_i}{|S|_{tot}} = \log_2 |S|_{tot} - \frac{1}{|S|_{tot}} \sum_{i=1}^{n} f_i \log_2 f_i$$
$$= \log_2(|S|_{tot}) - 2^{(\sum_{i=1}^{n} f_i \log_2 f_i - \log_2 |S|_{tot})}$$

**3** times of logarithmic computation and **1** time of exponential-function computation

- SOTA [1] needs thousands of table entries in TCAM for the logarithmic and exponential-function estimations to assure the relative error is under 1%
  - TCAM is expensive and power-hungry
  - SOTA needs a controller to populate the TCAM lookup tables in the switch

**Network traffic entropy estimation should avoid using TCAM to work entirely in programmable data planes**

[1] Sharma, N. K., Kaufmann et al. "Evaluating the power of flexible packet processing for network resource allocation" Symposium on Networked Systems Design and Implementation (NSDI 17) (pp. 67-82).

# Our contributions

1. **P4Log:** An algorithm for the estimation of logarithmic function
2. **P4Exp:** An algorithm for the estimation of exponential function
3. **P4Entropy:** A novel strategy allowing the estimation of network traffic entropy without relying on TCAM
4. We implemented the prototypes of the proposed algorithms and strategy in the P4 behavioral model [2], proving that they can be **entirely** executed in the programmable data plane.

[2] https://github.com/p4lang/behavioral-model

# P4Log algorithm

- **INPUT:** An L-bit integer $x$ (L $\in \{16, 32, 64\}$) and a given logarithmic base $d$
- **OUTPUT:** Estimation of $\log_d x \ll 10$ (i.e., $\log_d x \cdot 2^{10}$)

# P4Log algorithm

- **INPUT:** An L-bit integer $x$ (L $\in \{16, 32, 64\}$) and a given logarithmic base $d$
- **OUTPUT:** Estimation of $\log_d x \ll 10$ (i.e., $\log_d x \cdot 2^{10}$)

$$\log_d x = \log_2 x \cdot \log_d 2$$

$$\log_2 x \ll 10 = (\log_2 x)_{int} \ll 10 + (\log_2 x)_{dec} \ll 10$$

$$\log_d 2$$
(Constant)

# P4Log algorithm

- **INPUT:** An L-bit integer $x$ (L $\in \{16, 32, 64\}$) and a given logarithmic base $d$
- **OUTPUT:** Estimation of $\log_d x \ll 10$ (i.e., $\log_d x \cdot 2^{10}$)

$\log_d x$
$= \log_2 x \cdot \log_d 2$

$\log_2 x \ll 10$
$= (\log_2 x)_{int} \ll 10 + (\log_2 x)_{dec} \ll 10$

$\log_d 2$
(Constant)

index of leftmost 1 of $x$
(e.g., $(8)_{10} = (1000)_2$ and $\log_2 8 = 3$)

$n = (\log_2 x)_{int} \ll 10$

# P4Log algorithm

- **INPUT:** An L-bit integer $x$ (L $\in \{16, 32, 64\}$) and a given logarithmic base $d$
- **OUTPUT:** Estimation of $\log_d x \ll 10$ (i.e., $\log_d x \cdot 2^{10}$)

$\log_d x = \log_2 x \cdot \log_d 2$

$\log_2 x \ll 10 = (\log_2 x)_{int} \ll 10 + (\log_2 x)_{dec} \ll 10$

$\log_d 2$ (Constant)

index of leftmost 1 of $x$
(e.g., $(8)_{10} = (1000)_2$ and $\log_2 8 = 3$)

$n = (\log_2 x)_{int} \ll 10$

$(\log_2 x)_{dec} = \log_2(1 + \frac{x-2^n}{2^n})$

(e.g., $\log_2(1 + \frac{(110)_2-(100)_2}{\underbrace{(100)_2}_{N_{bits}=2}}) \ll 10 \approx \underbrace{0.585}_{N_{digits}=3} \ll 10$)

(e.g., $\log_2(1 + \frac{(1101)_2-(1000)_2}{\underbrace{(1000)_2}_{N_{bits=}=3}}) \ll 10 \approx \underbrace{0.7004}_{N_{digits}=4} \ll 10$)

Constant (Pre-computed: Binary search tree)

# P4Exp algorithm

- **INPUT:** An integer base $x$ and a real number exponent $d$
- **OUTPUT:** Estimation of $x^d$

# P4Exp algorithm

- **INPUT:** An integer base $x$ and a real number exponent $d$
- **OUTPUT:** Estimation of $x^d$

$$
\begin{aligned}
x^d &= 2^{d \log_2 x} \\
&= 2^{(d \log_2 x)_{int}} \cdot 2^{(d \log_2 x)_{dec}}
\end{aligned}
$$

$$2^{(d \log_2 x)_{int}} = 1 \ll ((d \log_2 x)_{int})$$

$$2^{(d \log_2 x)_{dec}}$$

# P4Exp algorithm

- **INPUT:** An integer base $x$ and a real number exponent $d$
- **OUTPUT:** Estimation of $x^d$

$$x^d = 2^{d \log_2 x} = 2^{(d \log_2 x)_{int}} \cdot 2^{(d \log_2 x)_{dec}}$$

$$2^{(d \log_2 x)_{int}} = 1 \ll ((d \log_2 x)_{int})$$

$$2^{(d \log_2 x)_{dec}}$$

- **Binomial series expansion**: $2^y = 1 + y + \frac{y(y-1)}{2!} + \frac{y(y-1)(y-2)}{3!} + \cdots$

# P4Exp algorithm

- **INPUT:** An integer base $x$ and a real number exponent $d$
- **OUTPUT:** Estimation of $x^d$

$$\boxed{\begin{aligned} x^d &= 2^{d \log_2 x} \\ &= 2^{(d \log_2 x)_{int}} \cdot 2^{(d \log_2 x)_{dec}} \end{aligned}}$$

$$\boxed{\begin{aligned} &2^{(d \log_2 x)_{int}} \\ &= 1 \ll ((d \log_2 x)_{int}) \end{aligned}} \qquad \boxed{2^{(d \log_2 x)_{dec}}}$$

- **Binomial series expansion**: $2^y = 1 + y + \frac{y(y-1)}{2!} + \frac{y(y-1)(y-2)}{3!} + \cdots$

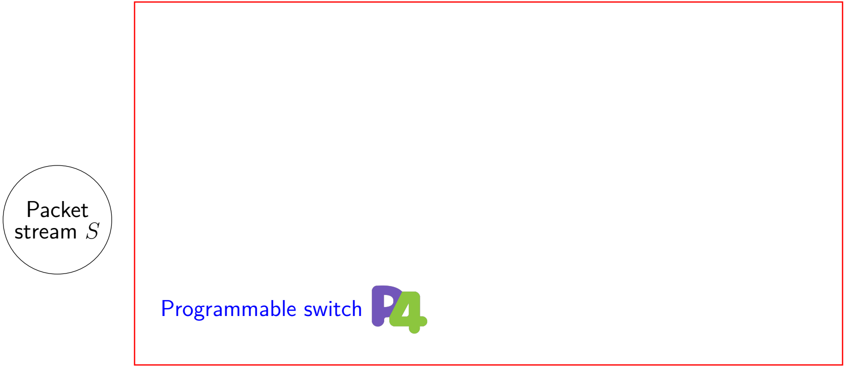
**So it holds that**:

$$2^{(d \log_2 x)_{dec}} = \underbrace{1 + (d \log_2 x)_{dec} + \frac{(d \log_2 x)_{dec}((d \log_2 x)_{dec} - 1)}{2!} + \cdots}_{N_{terms}}$$

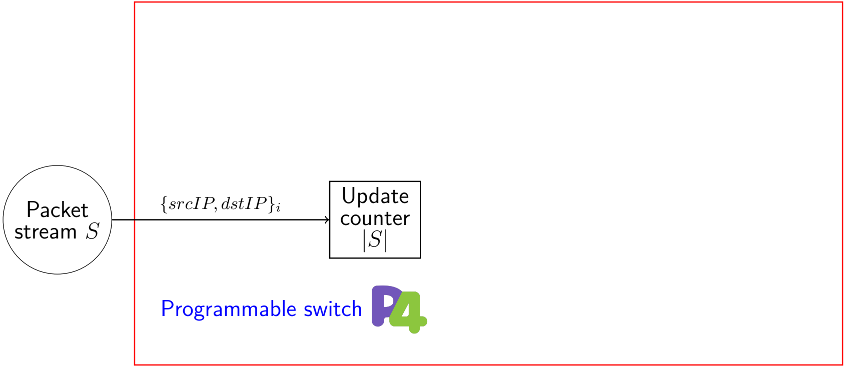
# P4Entropy strategy

Packet stream $S$

# P4Entropy strategy

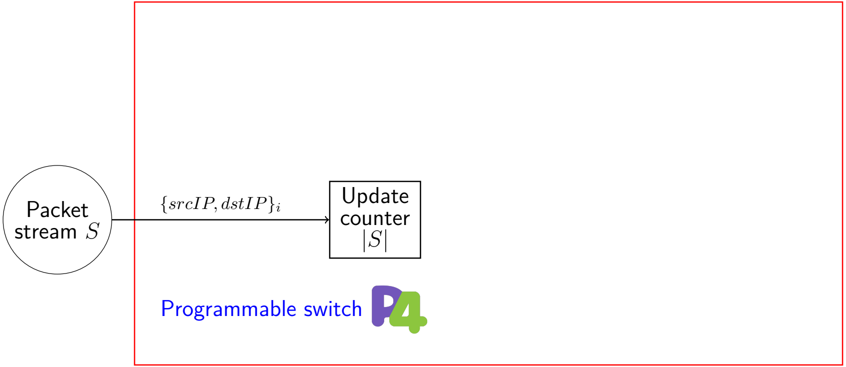


# P4Entropy strategy



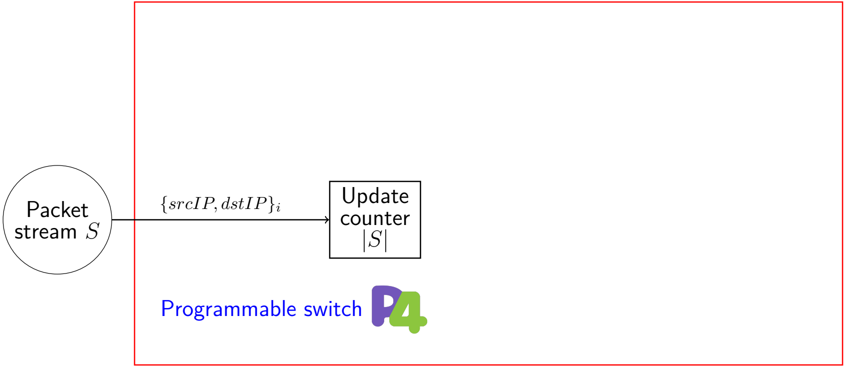

A new time interval starts

# P4Entropy strategy




The packets are identified by flow key $\{srcIP, dstIP\}$
The flow key can be any subset of 5 tuple without any loss of generality

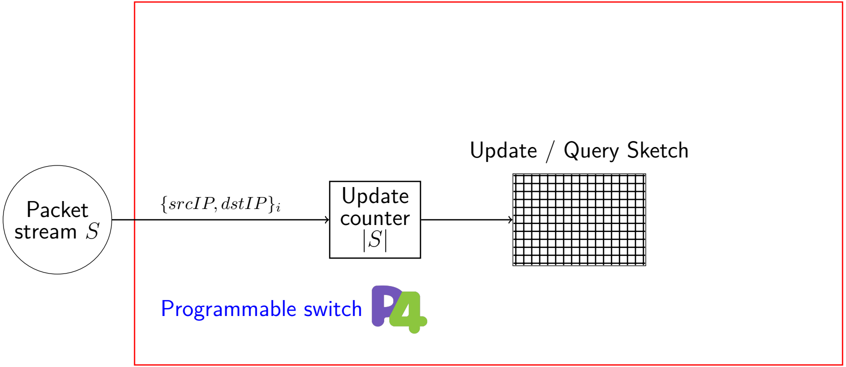
# P4Entropy strategy



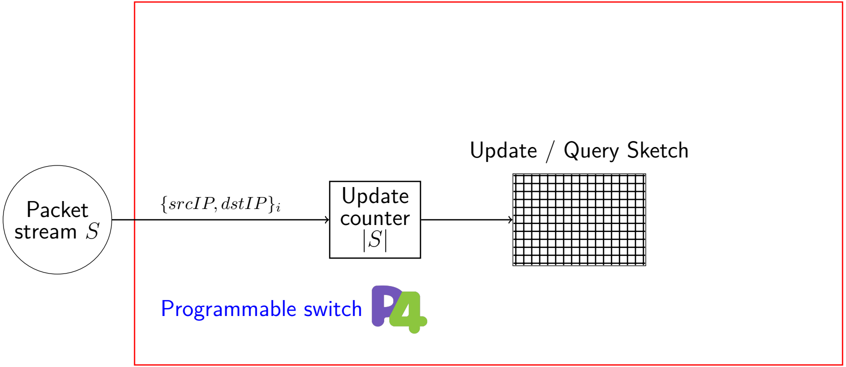

The packets are identified by flow key $\{srcIP, dstIP\}$
The flow key can be any subset of 5 tuple without any loss of generality

$$|S| = |S| + 1$$

# P4Entropy strategy



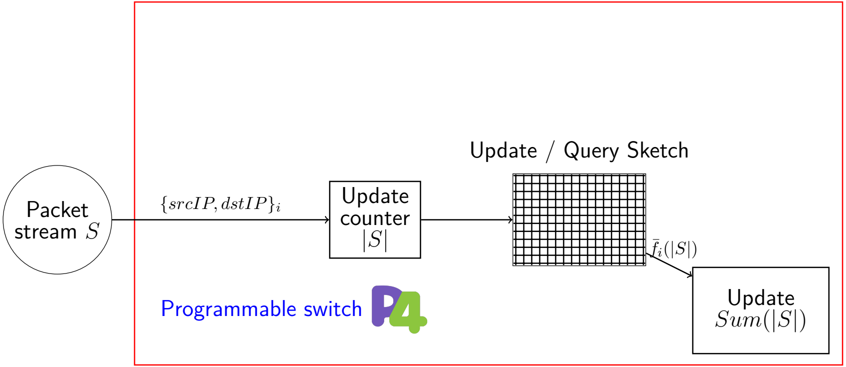
# P4Entropy strategy



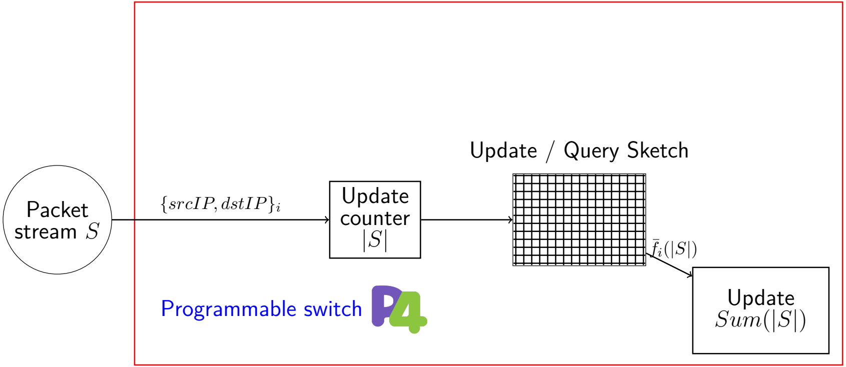

*Sketch* is a memory-efficient data structure to store and estimate the packet count $\overline{f}_i$ of each flow $i$.

- It is composed by $N_h$ hash functions with output size $N_s$.
- The accuracy of estimated packet count increases as $N_h$ or $N_s$ increases

# P4Entropy strategy

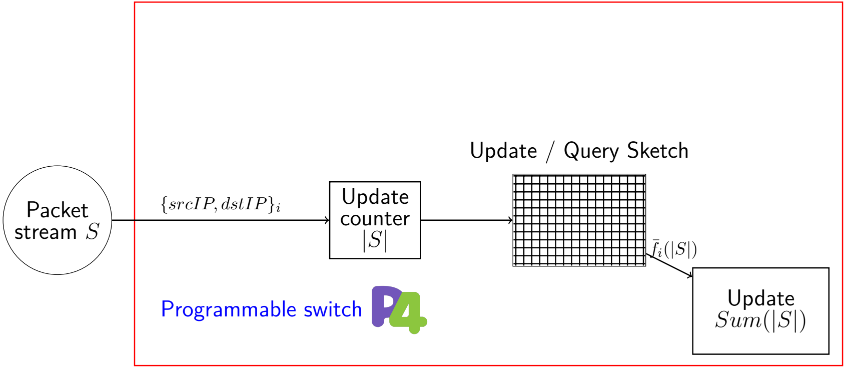


# P4Entropy strategy




$$Sum(|S|) = Sum(|S|-1) + \bar{f}_i(|S|)\log_2 \bar{f}_i(|S|) - (\bar{f}_i(|S|)-1)\log_2(\bar{f}_i(|S|)-1)$$

# P4Entropy strategy

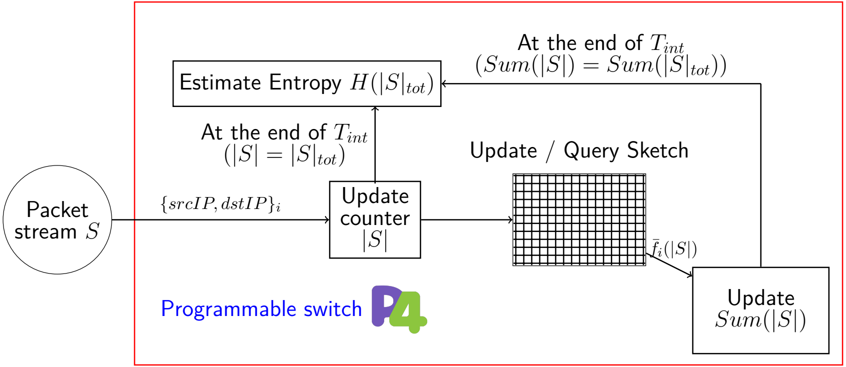



$$Sum(|S|) = Sum(|S|-1) + \bar{f}_i(|S|)\log_2 \bar{f}_i(|S|) - (\bar{f}_i(|S|)-1)\log_2(\bar{f}_i(|S|)-1)$$

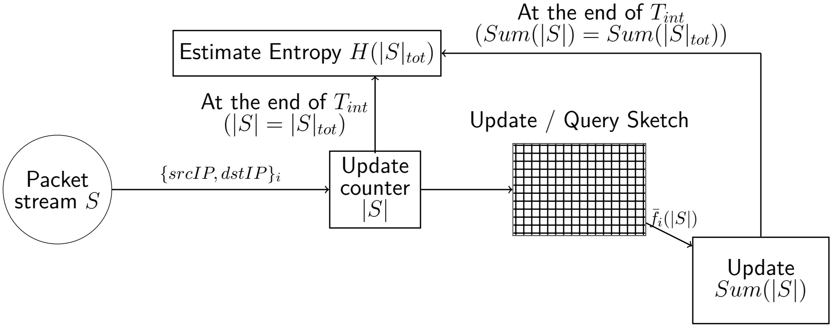
Applying L'Hopital's rule [3]: $Sum(|S|) = Sum(|S|-1) + \log_2 \bar{f}_i(|S|) + \frac{1}{\ln 2}$

[3] D. J. Struik, "The origin of L'Hopital's rule," The Mathematics Teacher, vol. 56, no. 4, pp. 257–260, 1963.

# P4Entropy strategy

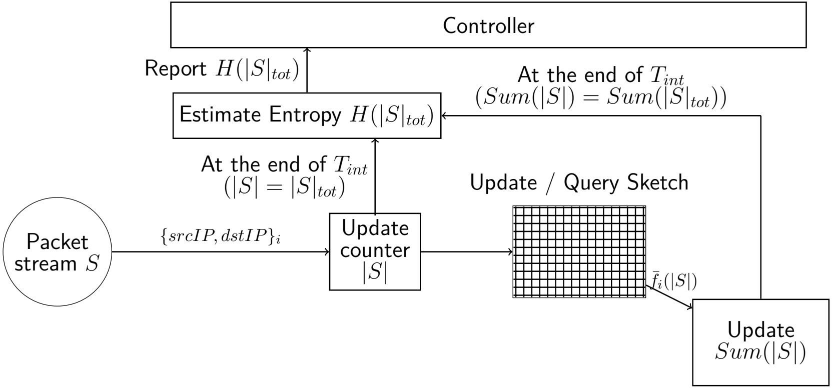


# P4Entropy strategy

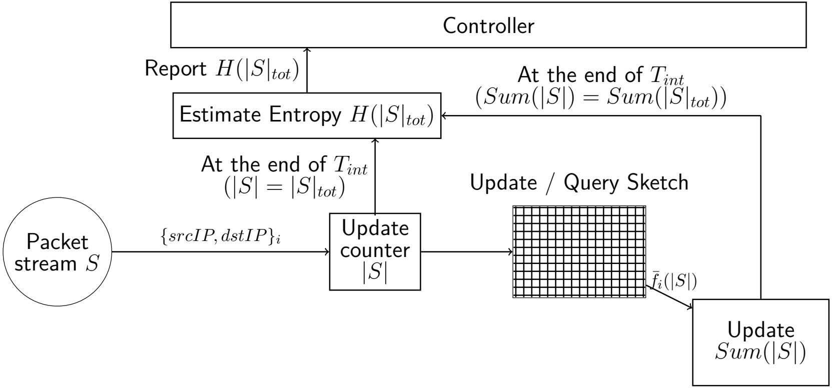



$$H(|S|_{tot}) = \underbrace{\log_2(|S|_{tot})}_{P4Log} - \underbrace{2^{\overbrace{\log_2(Sum(|S|_{tot}))}^{P4Log} - \overbrace{\log_2(|S|_{tot})}^{P4Log}}}_{P4Exp}$$

# P4Entropy strategy



# P4Entropy strategy



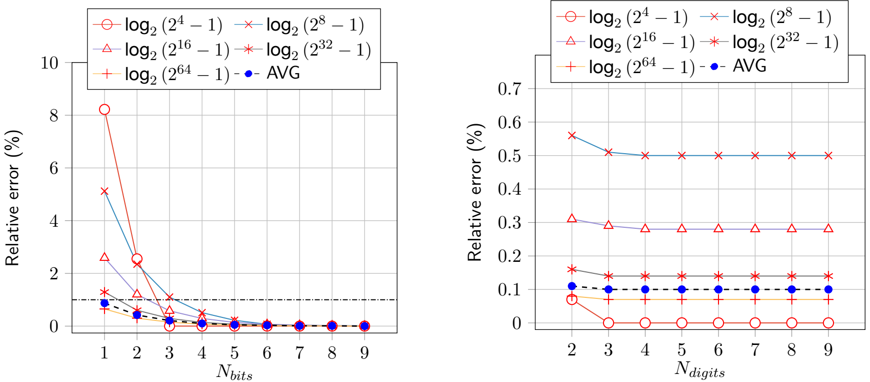

**P4Entropy can estimate network traffic entropy entirely in the switch**

# Evaluation settings

Metric: *Relative error*

- **P4Log** Given an input value $x$, relative error is defined as $\frac{|P4Log(x,2)-\log_2 x|}{\log_2 x} \cdot 100\%$, where $\log_2 x$ is the exact value.
- **P4Exp** Given an input base $x$ and an exponent $d$, the relative error is defined as $\frac{\left|P4Exp(x,d)-x^d\right|}{x^d} \cdot 100\%$.
- **P4Entropy** We call $\hat{H}$ the estimated traffic entropy in an observation window and $H$ its exact value. The relative error is defined as the average value of $\frac{\left|H-\hat{H}\right|}{H} \cdot 100\%$ in the 10 observation windows each composed by $2^{21}$ packets captured from a real flow trace [3]

[3] CAIDA UCSD Anonymized Internet Traces Dataset `http://www.caida.org/data/passive/passive_dataset.xml`



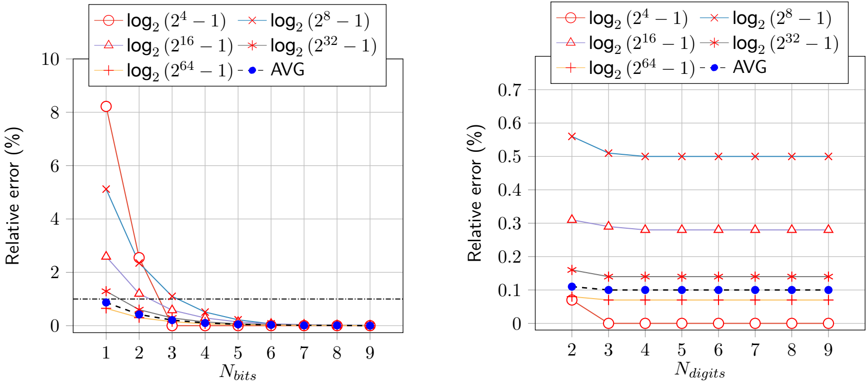

(a) Sensitivity to $N_{bits}$ ($N_{digits} = 3$) (b) Sensitivity to $N_{digits}$ ($N_{bits} = 4$)

Figure: Sensitivity of P4Log to $N_{bits}$ (a) and $N_{digits}$ (b)

$$\log_2(1 + \frac{(110)_2-(100)_2}{\underbrace{(100)_2}_{N_{bits}=2}}) \ll 10 \approx \underbrace{0.585}_{N_{digits}=3} \ll 10$$

$$2^l - 1 = \underbrace{1\cdots1}_{l}(l \in 4, 8, 16, 32, 64)$$

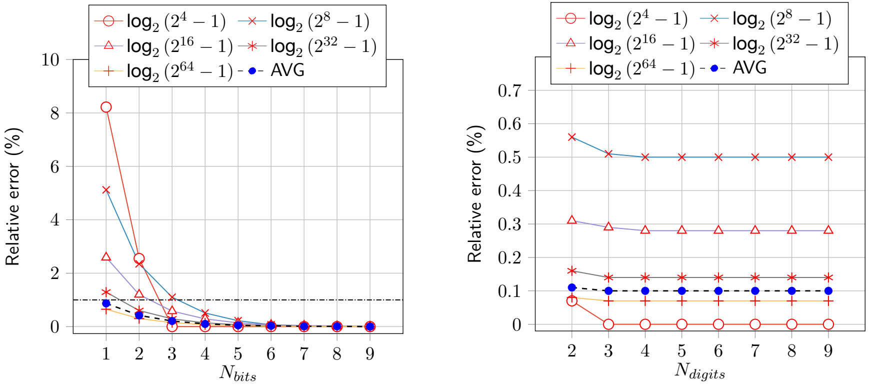
# Results




(a) Sensitivity to $N_{bits}$ ($N_{digits} = 3$)　(b) Sensitivity to $N_{digits}$ ($N_{bits} = 4$)

Figure: Sensitivity of P4Log to $N_{bits}$ (a) and $N_{digits}$ (b)

All the bits after $N_{bits}$ are ignored and considered as 0. The algorithm leads to worst-case estimations when most significant bits after $N_{bits}$ are 1s.

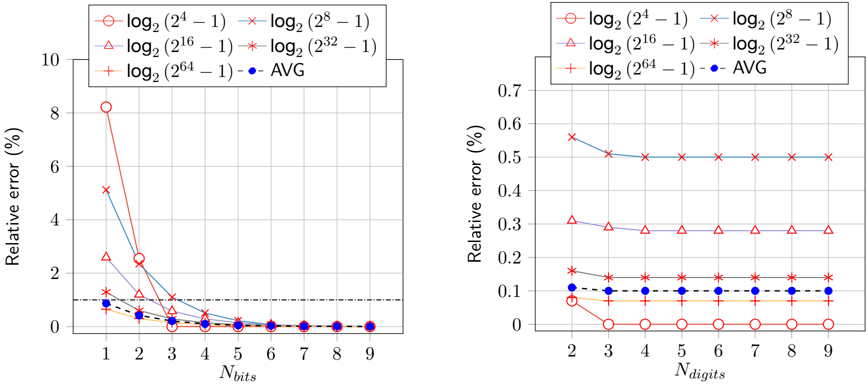



(a) Sensitivity to $N_{bits}$ ($N_{digits} = 3$) (b) Sensitivity to $N_{digits}$ ($N_{bits} = 4$)

Figure: Sensitivity of P4Log to $N_{bits}$ (a) and $N_{digits}$ (b)

**AVG** is the average relative error in logarithm estimations among randomly-selected $5 \cdot 10^6$ integer numbers such that $x \in \{1, 2^{64} - 1\}$

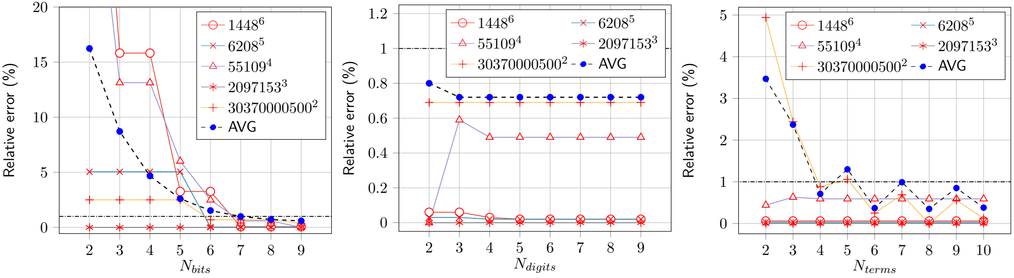
# Results




(a) Sensitivity to $N_{bits}$ ($N_{digits} = 3$) (b) Sensitivity to $N_{digits}$ ($N_{bits} = 4$)

Figure: Sensitivity of P4Log to $N_{bits}$ (a) and $N_{digits}$ (b)

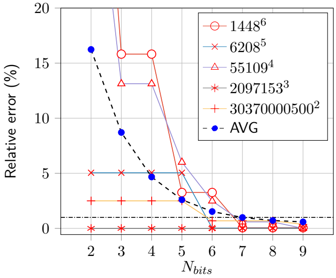
Relative error $< 1\%$ when $N_{bits} = 4$ and $N_{digits} = 3$



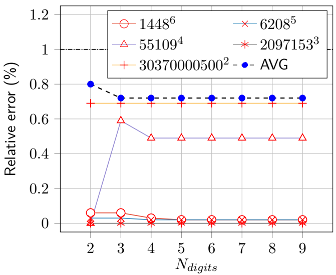

(a) $N_{bits}$ (b) $N_{digits}$ (c) $N_{terms}$

Figure: Sensitivity analysis of P4Exp ( $N_{bits} = 7$, $N_{digits} = 3$ and $N_{terms} = 7$)

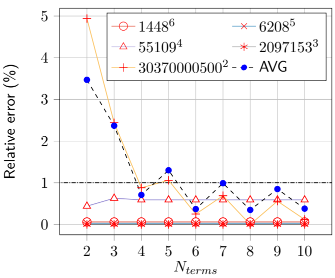
We fix the integer exponent $d$ to a chosen value, then we find the largest 64-bit integer base $x$ that maximizes the output $x^d$ within 64 bits




(a) $N_{bits}$



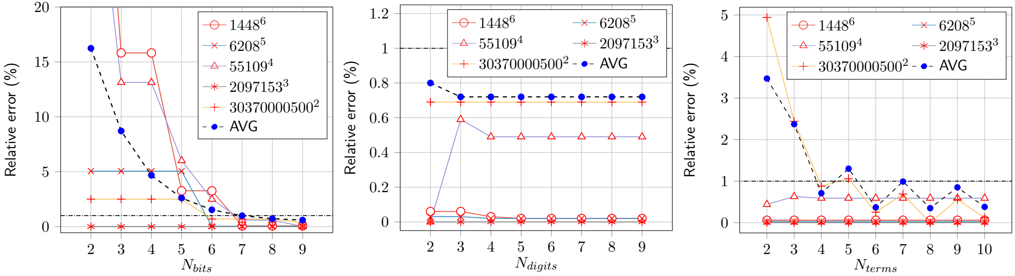

(b) $N_{digits}$




(c) $N_{terms}$

Figure: Sensitivity analysis of P4Exp ( $N_{bits} = 7$, $N_{digits} = 3$ and $N_{terms} = 7$)

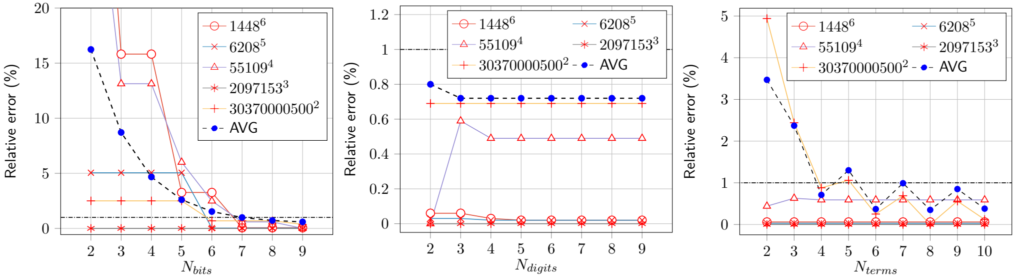
**AVG** is the average relative error in exponential function estimation among $5 \cdot 10^6$ integer numbers with base $x \in \{1, 2^{32} - 1\}$ and exponent $d \in \{2, 32\}$ both randomly chosen




(a) $N_{bits}$ (b) $N_{digits}$ (c) $N_{terms}$

Figure: Sensitivity analysis of P4Exp ( $N_{bits} = 7$, $N_{digits} = 3$ and $N_{terms} = 7$)

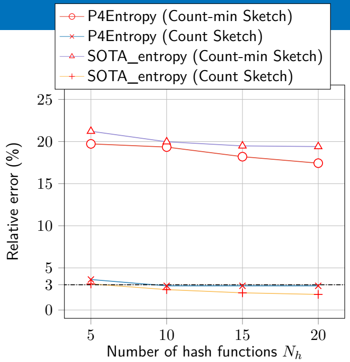
$$2^{(d\log_2 x)_{dec}} = \underbrace{1 + (d\log_2 x)_{dec} + \frac{(d\log_2 x)_{dec}((d\log_2 x)_{dec} - 1)}{2!} + \cdots}_{N_{terms}}$$



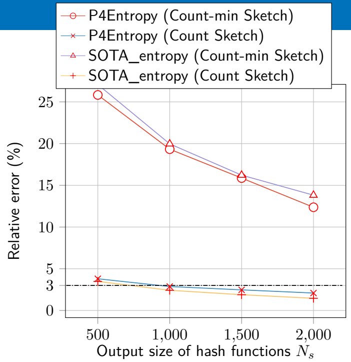

(a) $N_{bits}$ (b) $N_{digits}$ (c) $N_{terms}$

Figure: Sensitivity analysis of P4Exp ( $N_{bits} = 7$, $N_{digits} = 3$ and $N_{terms} = 7$)

Relative error $< 1\%$ when $N_{bits} = 7$, $N_{digits} = 3$ and $N_{terms} = 7$




(a) Sensitivity to $N_h$ ($N_s = 1000$)

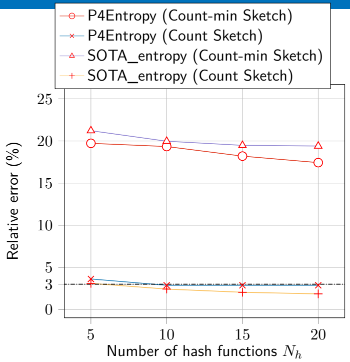



(b) Sensitivity to $N_s$ ($N_h = 10$)

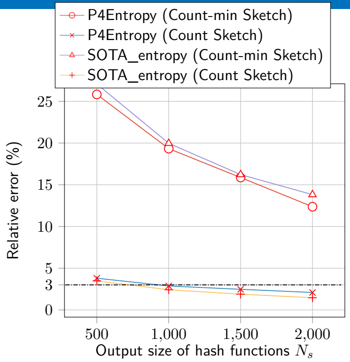
Figure: Sensitivity of P4Entropy to sketch size ($N_{bits} = 4$, $N_{digits} = 3$ and $N_{terms} = 7$)

Network traffic entropy estimation is more sensitive to the overestimation caused by **Count-min Sketch** with respect to **Count Sketch**

**SOTA_entropy**: Lapolli, Ângelo Cardoso, Jonatas Adilson Marques, and Luciano Paschoal Gaspary. "Offloading real-time ddos attack detection to programmable data planes." 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM). IEEE, 2019.




(a) Sensitivity to $N_h$ ($N_s = 1000$)




(b) Sensitivity to $N_s$ ($N_h = 10$)

Figure: Sensitivity of P4Entropy to sketch size ($N_{bits} = 4$, $N_{digits} = 3$ and $N_{terms} = 7$)

3% is the maximum possible relative error ensuring that accuracy of practical monitoring applications is not affected [4]

[4] Ashwin Lall et al. "Data streaming algorithms for estimating entropy of network traffic". In: ACM SIGMETRICS Performance Evaluation Review. Vol. 34. 1. pp. 145-156, 2006.

# Conclusion and future work

1. We presented two new algorithms, **P4Log** and **P4Exp**, providing the estimation of logarithm and exponential function in P4.
2. Based on these two algorithms, we also proposed **P4Entropy** strategy to estimate the network traffic entropy.
3. The algorithms have the similar accuracy to the state-of-the-art solutions but do not rely on expensive and energy-hungry TCAMs while working entirely in the switch.
4. P4Entropy adopts a time-interval-based observation window that may allow the controller to synchronize the entropy collected from all switches

**Future work:**

1. Implement network traffic entropy-based DDoS detection entirely in programmable data planes
2. Test proposed algorithms and strategy on a real testbed