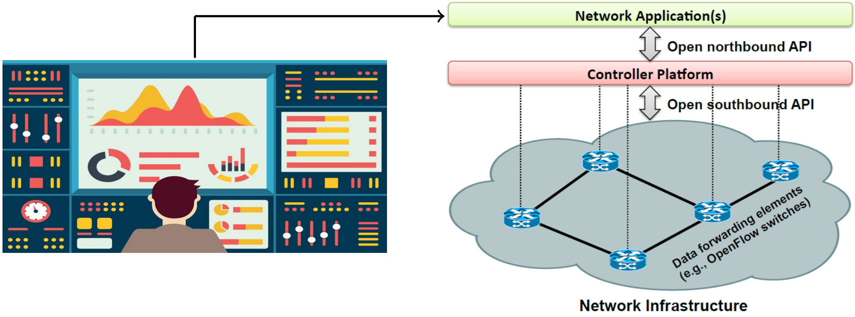
Incremental Deployment of Programmable Switches for Network-wide Heavy-Hitter Detection

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

[1] *FBK CREATE-NET Research Center, Trento, Italy*
[2] *University of Bologna, Bologna, Italy*
[3] *Queen Mary University of London, London, UK*

IEEE NetSoft
25th June, 2019

Network monitoring in Software Defined Networks

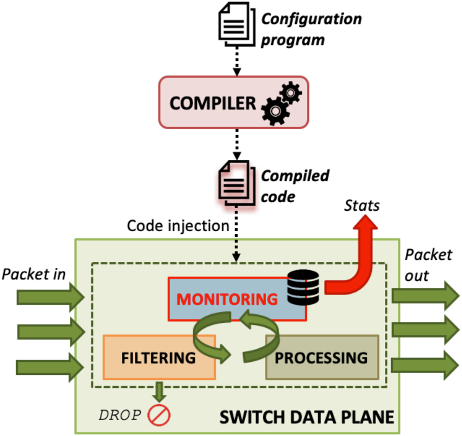


Drawbacks for network monitoring in today's SDN:

- Need to know which flows to monitor in advance
- Complicated processing
- Large communication overhead

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

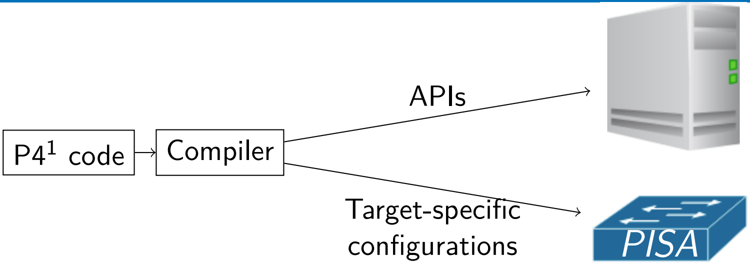
Programmable data plane for monitoring



- Enables the implementation of new network monitoring solutions directly in switch hardware

Programmable data plane

Example



Existing PISA-supported programmable switches:[2]

- Barefoot's Tofino switches
- Intel FlexPipe
- Cavium XPliant switches
- Texas Instruments' Reconfigurable Match Tables

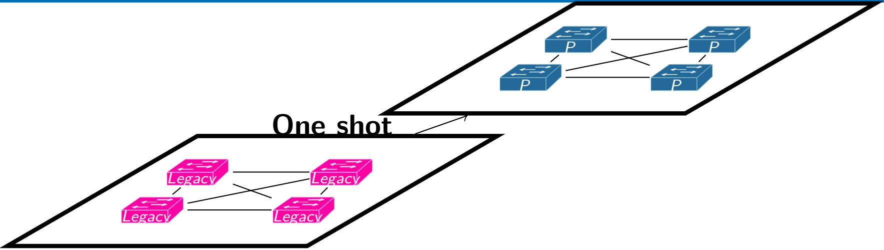
[1] Bosshart, Pat, et al. "P4: Programming protocol-independent packet processors." ACM SIGCOMM Computer Communication Review 44.3 (2014): 87-95.

[2] 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).

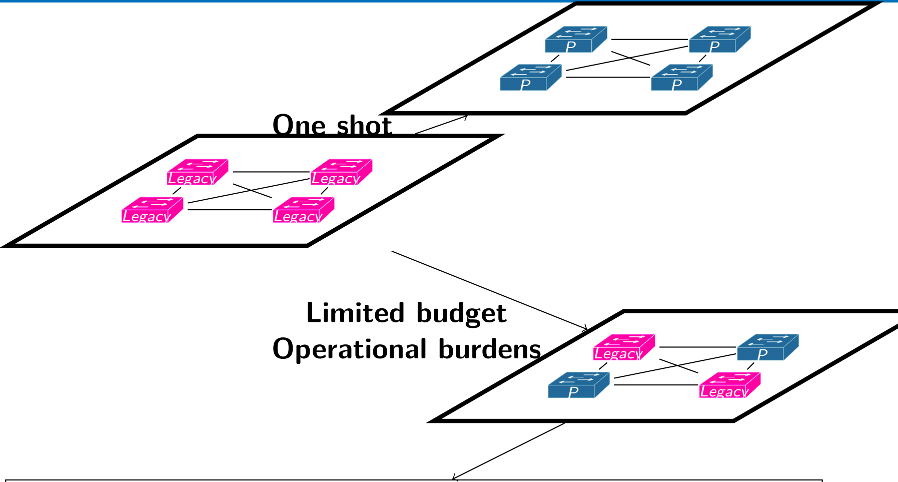
Our contributions

1. A novel **incremental deployment algorithm** for the placement of programmable switches and simultaneously maximize the network monitoring operation
2. A new **strategy for network-wide heavy-hitter detection** which has better performance than the state of the art while deploying programmable switches through our incremental deployment algorithm

Incremental deployment — Problem to solve



Incremental deployment Problem to solve



One shot

Limited budget
Operational burdens

How to obtain the best performance in network monitoring?

Incremental deployment

Idea

Given the possibility to upgrade a limited number of legacy switches to programmable switches, replace those that monitor the most **distinct flows** to obtain **highest flow visibility** of network

Incremental deployment

Idea

Given the possibility to upgrade a limited number of legacy switches to programmable switches, replace those that monitor the most **distinct flows** to obtain **highest flow visibility** of network

Distinct flow: A set of packets identified by the same pattern, named *flow key* (e.g., srcIP, dstIP, src port, dst port, protocol)

Incremental deployment

Idea

Given the possibility to upgrade a limited number of legacy switches to programmable switches, replace those that monitor the most **distinct flows** to obtain **highest flow visibility** of network

Distinct flow: A set of packets identified by the same pattern, named *flow key* (e.g., srcIP, dstIP, src port, dst port, protocol)

Flow key

Figure copyright: Angry birds owned by Rovio Entertainment Ltd

Incremental deployment

Idea

Given the possibility to upgrade a limited number of legacy switches to programmable switches, replace those that monitor the most **distinct flows** to obtain **highest flow visibility** of network

Distinct flow: A set of packets identified by the same pattern, named *flow key* (e.g., srcIP, dstIP, src port, dst port, protocol)

Flow key

Distinct flow

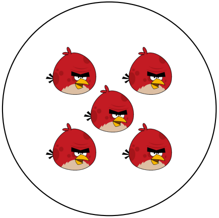
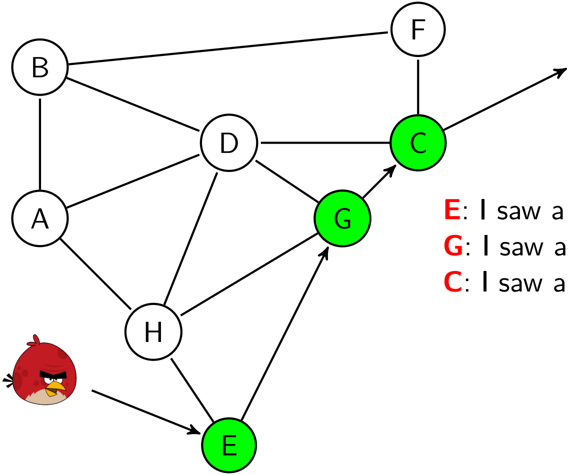


Figure copyright: Angry birds owned by Rovio Entertainment Ltd

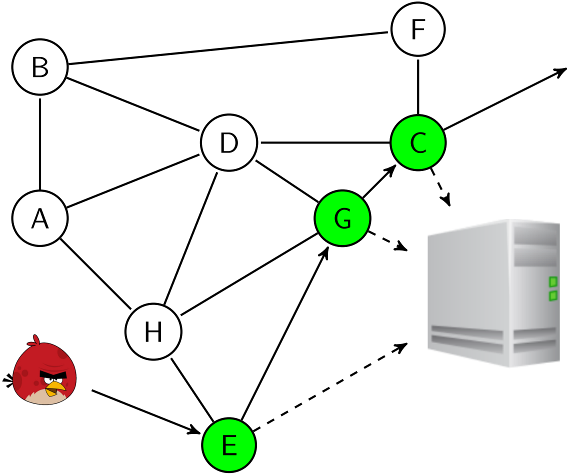
Why monitor distinct flows?



E: I saw a flow!
G: I saw a flow!
C: I saw a flow!

How many flows are there?

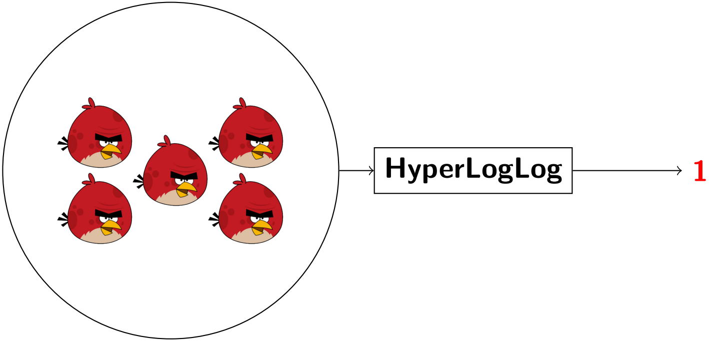
Why monitor distinct flows?



E, G, C: We monitored **three** times the same flow!

Controller: Must be realized that there is only **one** distinct flow!

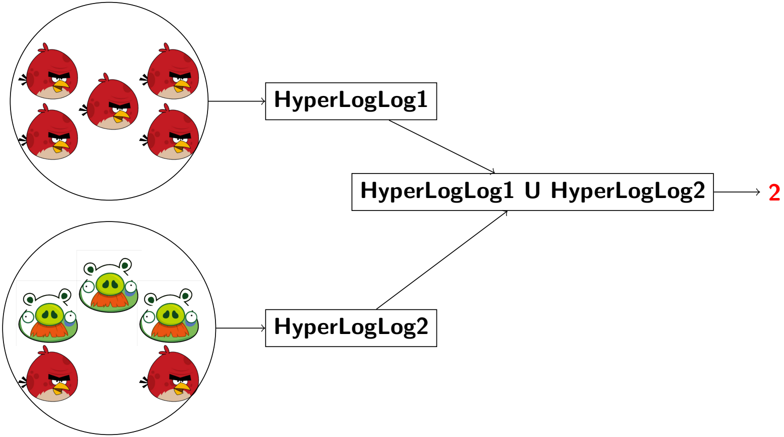
How to count distinct flows HyperLogLog



- **A fast estimation algorithm:** Time complexity is only O(1)
- **Efficient and accurate:** 1690 bytes can estimate 10^9 numbers with standard error 2%.[3]

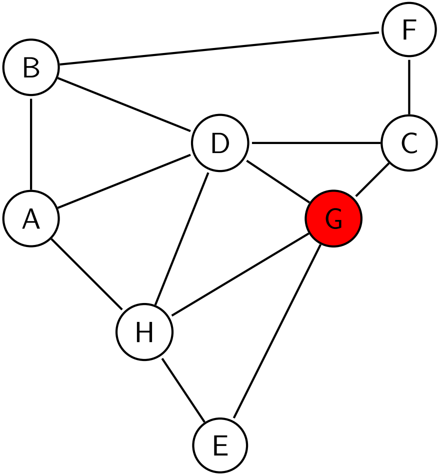
[3] Flajolet, Philippe, et al. "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm." Discrete Mathematics and Theoretical Computer Science. Discrete Mathematics and Theoretical Computer Science, 2007.

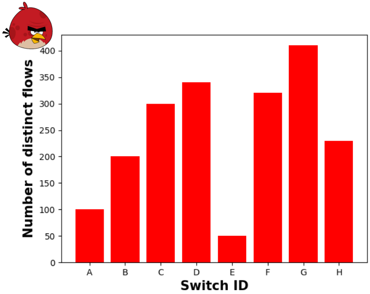
How to count distinct flows HyperLogLog



- **Simple in estimating the union of HyperLogLogs:** Merge HyperLogLog data structures to get a new HyperLogLog data structure and obtain the count of the resulting set.

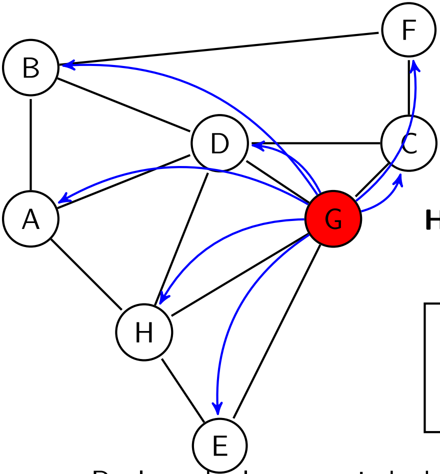
Incremental deployment algorithm

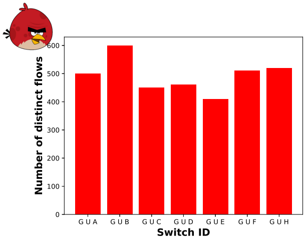




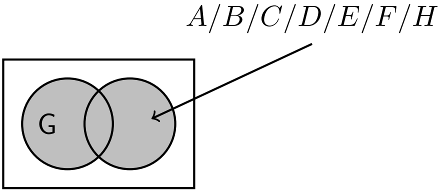
Replace the legacy switch that is crossed by **the largest number of distinct flows** among all switches to programmable switch

Incremental deployment algorithm



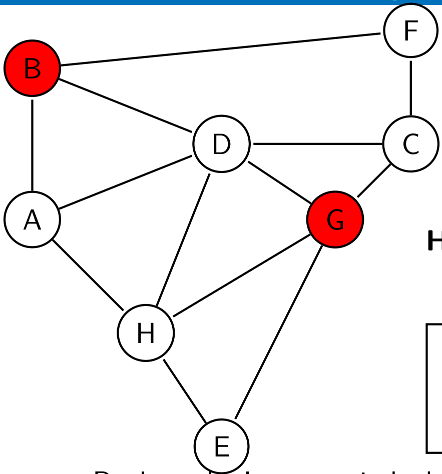


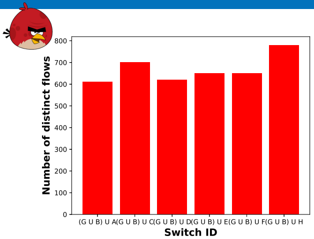
HyperLogLog



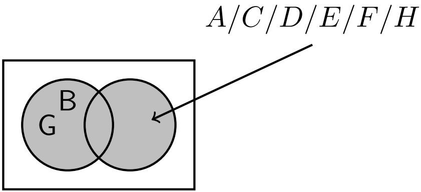
Replace the legacy switch that leads to **the largest union of distinct flows** with previously deployed programmable switch

Incremental deployment algorithm





HyperLogLog



Replace the legacy switch that leads to **the largest union of distinct flows** with all previously deployed programmable switches, and **iterate** until the maximum number of programmable switches to be replaced

Heavy-hitter detection

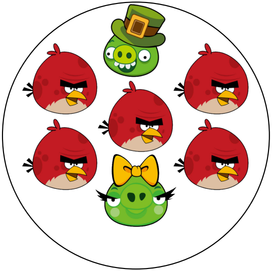
We decide to investigate **heavy-hitter detection** as monitoring use case to evaluate our Incremental deployment algorithm

Heavy-hitter detection

We decide to investigate **heavy-hitter detection** as monitoring use case to evaluate our Incremental deployment algorithm

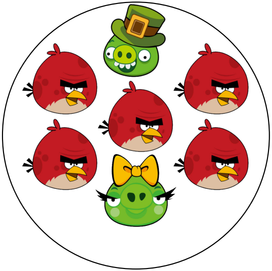
Heavy-hitter detection

We decide to investigate **heavy-hitter detection** as monitoring use case to evaluate our Incremental deployment algorithm



Heavy-hitter detection

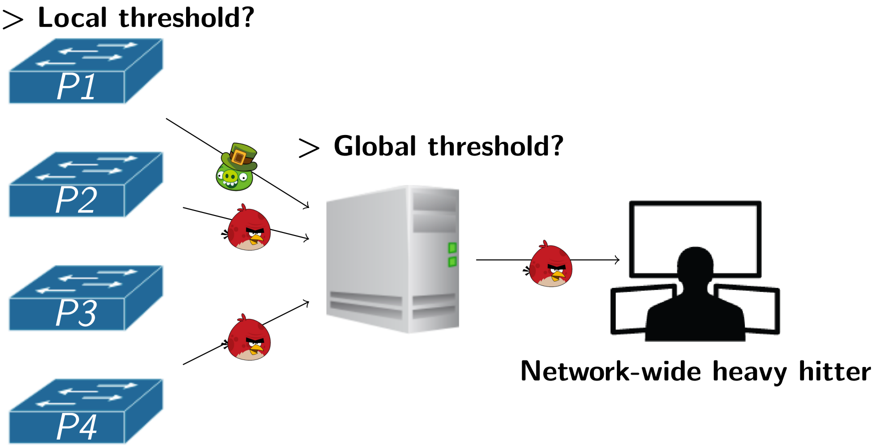
We decide to investigate **heavy-hitter detection** as monitoring use case to evaluate our Incremental deployment algorithm



Heavy hitter detection

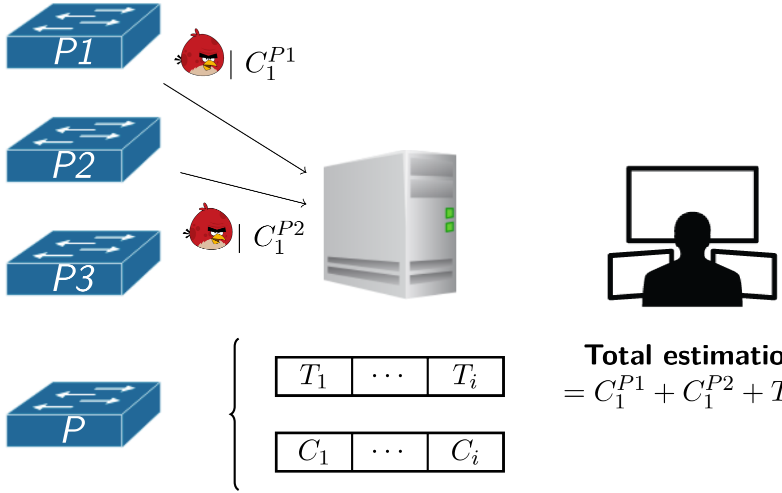
identifies the flows that consume more than **a fraction of the total generated packets (i.e., threshold)** in a given time interval

Network-wide Heavy-hitter detection



Network-wide Heavy-hitter detection

> Local threshold?



T_1	$\cdots$	T_i

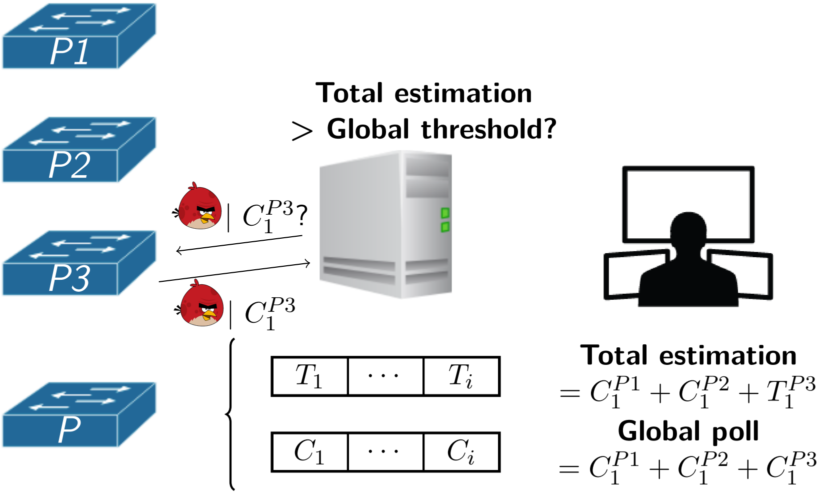
C_1	$\cdots$	C_i

Total estimation

$$= C_1^{P1} + C_1^{P2} + T_1^{P3}$$

Harrison, Rob, et al. "Network-Wide Heavy Hitter Detection with Commodity Switches." Proceedings of the Symposium on SDN Research. ACM, 2018.

Network-wide Heavy-hitter detection



Total estimation
$>$ **Global threshold?**

T_1	$\cdots$	T_i
C_1	$\cdots$	C_i

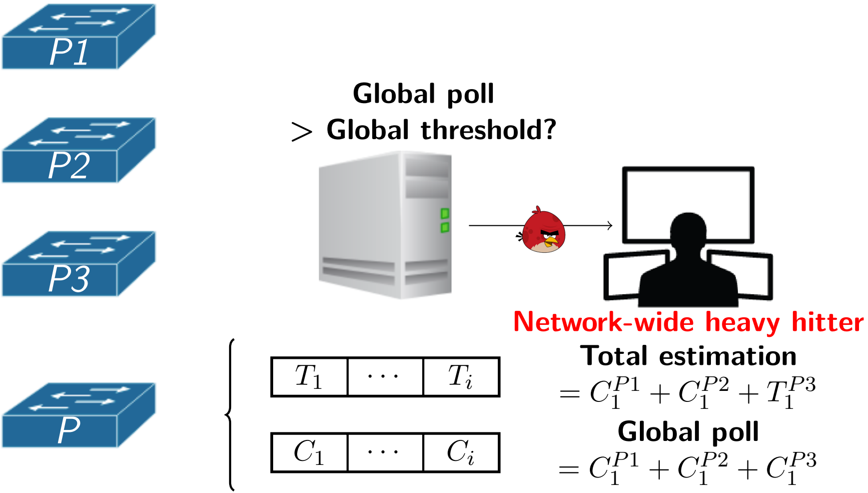
Total estimation
$= C_1^{P1} + C_1^{P2} + T_1^{P3}$

Global poll
$= C_1^{P1} + C_1^{P2} + C_1^{P3}$

Harrison, Rob, et al. "Network-Wide Heavy Hitter Detection with Commodity Switches." Proceedings of the Symposium on SDN Research. ACM, 2018.

Network-wide Heavy-hitter detection

SOTA



Global poll
$>$ **Global threshold?**

Network-wide heavy hitter

T_1	$\cdots$	T_i

C_1	$\cdots$	C_i

Total estimation

$$= C_1^{P1} + C_1^{P2} + T_1^{P3}$$

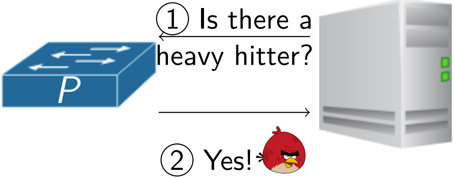
Global poll

$$= C_1^{P1} + C_1^{P2} + C_1^{P3}$$

Harrison, Rob, et al. "Network-Wide Heavy Hitter Detection with Commodity Switches." Proceedings of the Symposium on SDN Research. ACM, 2018.

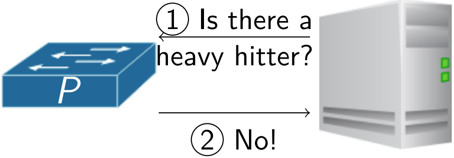
False positive and False negative

False positive



Detected is not a true heavy hitter!

False negative



There exists a heavy hitter !

Higher F1 score means **less** wrongly detected (False positive) and undetected heavy hitters (False negative)

SOTA in an incremental deployment scenario

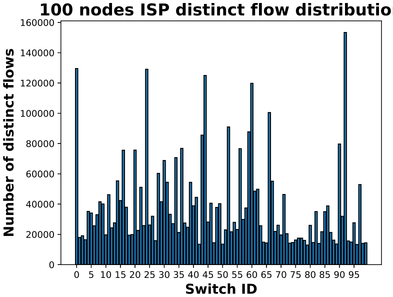


Figure: Number of distinct flows crossing each switch

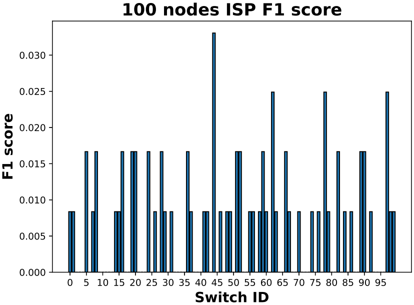


Figure: F1 score of SOTA with single programmable-switch deployment

F1 score of SOTA has **weak correlation** with number of distinct flows

SOTA in an incremental deployment scenario

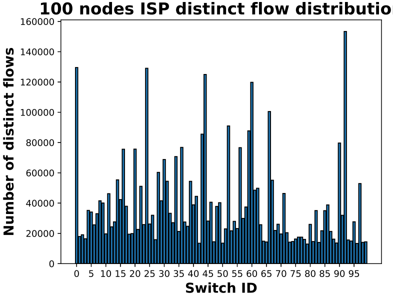


Figure: Number of distinct flows crossing each switch

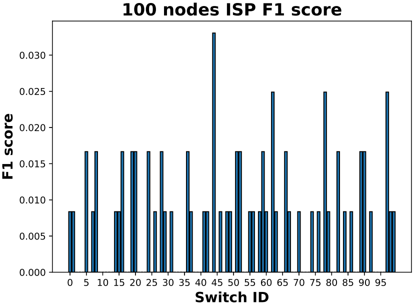
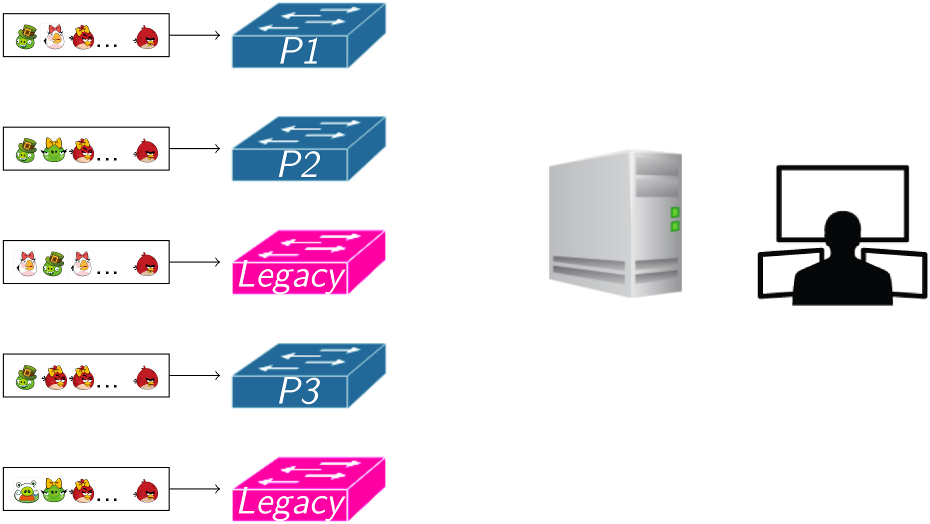


Figure: F1 score of SOTA with single programmable-switch deployment

F1 score of SOTA has **weak correlation** with number of distinct flows

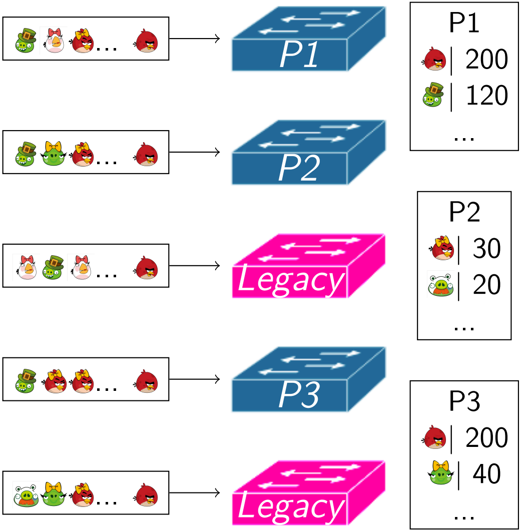
→ **NWHHD+**

NWHHD+ in an incremental deployment scenario



NWHHD+ in an incremental deployment scenario

> Sampling threshold & min?



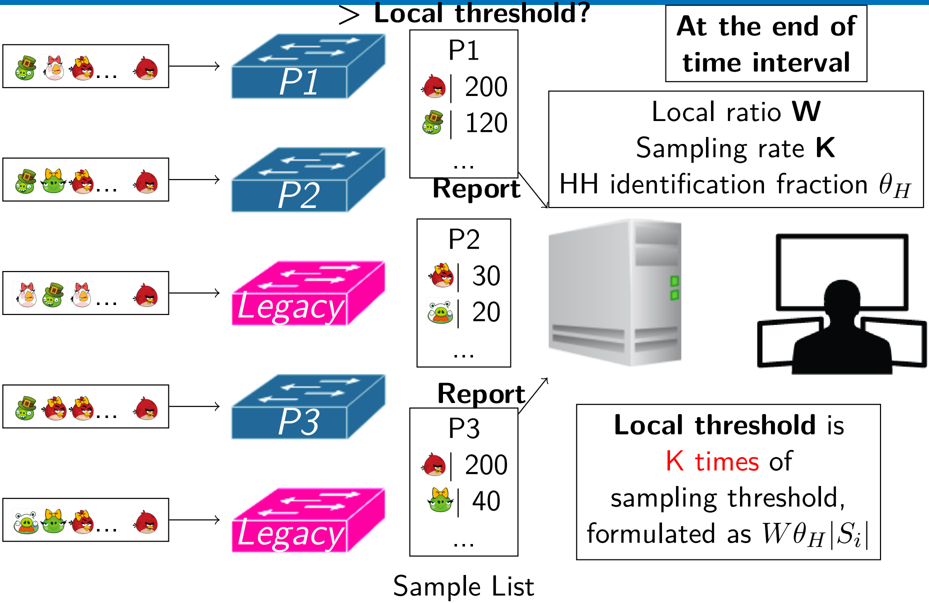
Sample List

During time interval

Local ratio **W**
Sampling rate **K**
HH identification fraction θ_H

Sampling threshold is a given fraction $\frac{W\theta_H}{K}$ of incoming packets $|S_i|$ in each switch

NWHHD+ in an incremental deployment scenario



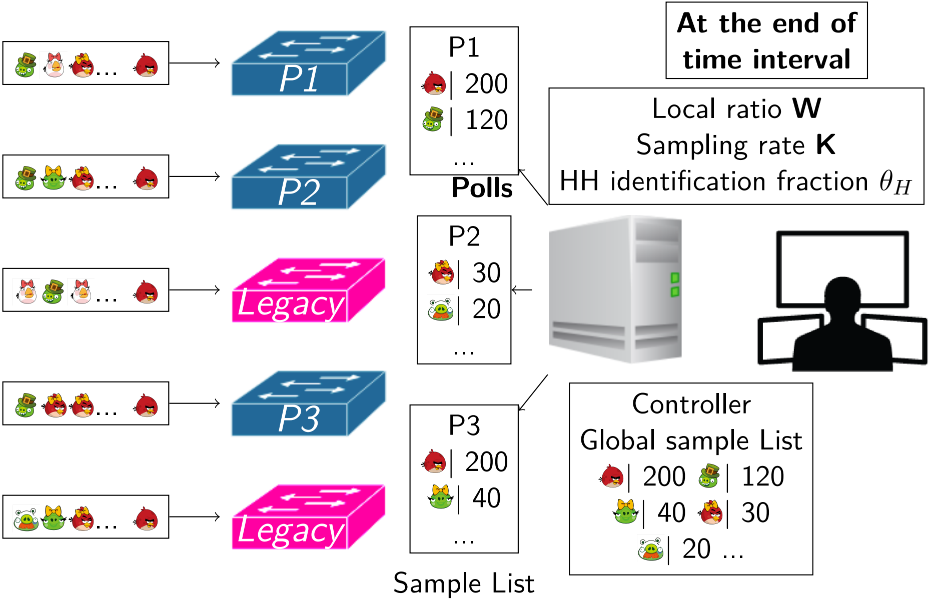
At the end of time interval

Local ratio **W**
Sampling rate **K**
HH identification fraction θ_H

Local threshold is K times of sampling threshold, formulated as $W\theta_H|S_i|$

Sample List

NWHHD+ in an incremental deployment scenario



At the end of time interval

Local ratio **W**
Sampling rate **K**
HH identification fraction θ_H

Polls

P1
| 200
| 120
...

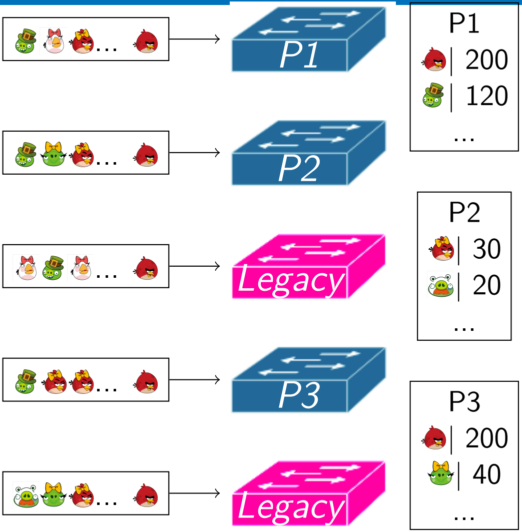
P2
| 30
| 20
...

P3
| 200
| 40
...

Sample List

Controller
Global sample List
| 200 | 120
| 40 | 30
| 20 ...

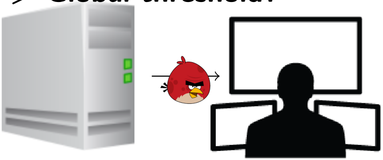
NWHHD+ in an incremental deployment scenario



At the end of time interval

Local ratio **W** Sampling rate **K**
HH identification fraction θ_H

$>$ **Global threshold?**



Controller
Global sample List
| 200 | 120
| 40 | 30
| 20 ...

Global threshold is the fraction θ_H of total packets in Global sample list $|S_{tot}|$

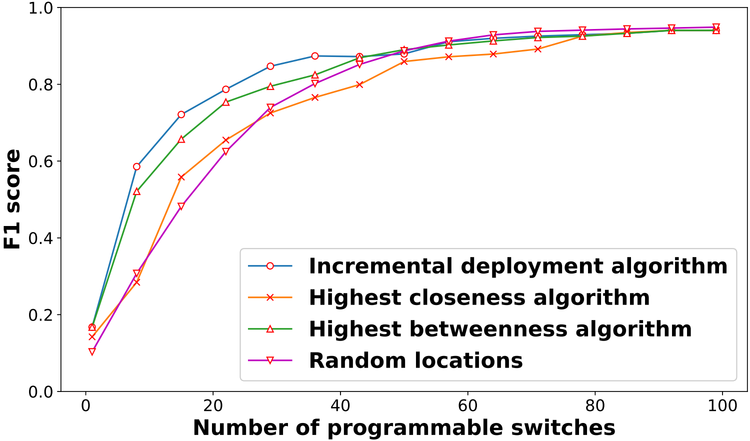
Evaluation settings

- **Flow trace:** 2018-passive CAIDA flow trace
- **Time interval:** 5s
- **# packets per time interval:** Around 2.3 million packets
- **Network topology:** A 100-nodes ISP backbone network
- Local ratio **W:** 1
- Sampling rate **K:** 10
- Heavy-hitter identification fraction θ_H: 0.05%
- Local minimum **min:** 1

Results

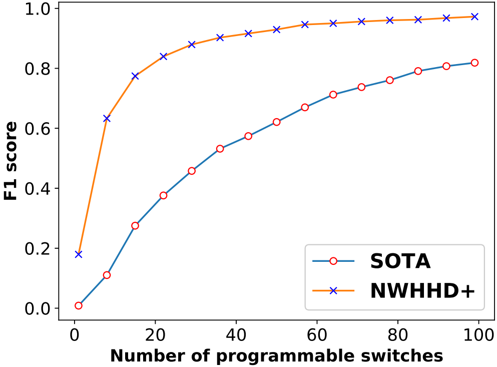
NWHHD+:

Incremental deployment algorithm VS. Three existing algorithms



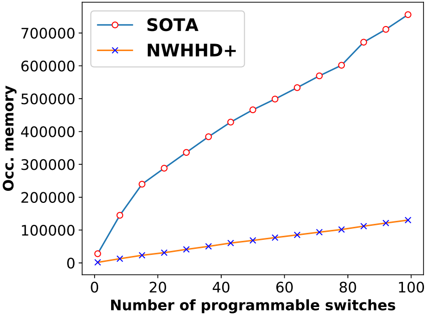
Results

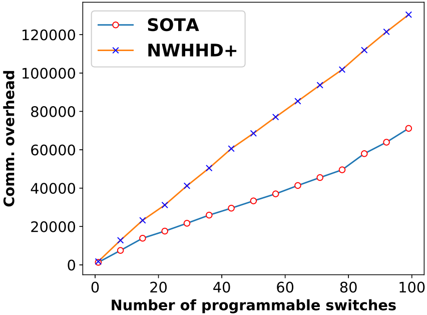
Incremental deployment scenario:
NWHHD+ VS. **SOTA**[4]



[4] Harrison, Rob, et al. "Network-Wide Heavy Hitter Detection with Commodity Switches." Proceedings of the Symposium on SDN Research. ACM, 2018.

Results





- **Y-axis unit**: {flow key, packet count #} pair

Results Sensitivity analysis

Evaluated metrics	SOTA	NWHHD+			
		K=1.2	K=10	K=20	K=100
F1 score	0.821	0.846	0.948	0.970	0.998
Communication overhead	71877	24760	131707	218370	570956
Occupied memory	760042	24661	131608	218264	570875

Table: Sensitivity to K in the case of full deployment $(W = 1)$

Evaluated metrics	SOTA	NWHHD+			
		W=1	W=3	W=5	W=20
F1 score	0.821	0.948	0.907	0.881	0.823
Communication overhead	71877	131707	60354	41076	13898
Occupied memory	760042	131608	60255	40977	13799

Table: Sensitivity to W in the case of full deployment $(K = 10)$

Conclusion and future work

1. Designed an Incremental deployment algorithm for the purpose of network flow monitoring, aiming at maximizing the flow visibility with limited number of programmable switches
2. Proposed a new strategy for network-wide heavy hitter detection in ISP backbone networks
   - Has better trade-off on accuracy of heavy hitter detection, memory occupation and communication overhead when adopted jointly with our Incremental deployment algorithm than the state of the art

Future work:

1. Extend our current work to a wider range of network monitoring tasks, such as entropy estimation, heavy-change detection and DDoS detection
2. Implement proposed algorithms in P4 language and test them on real testbed