HH-IPG: Leveraging Inter-Packet Gap Metrics in P4 Hardware for Heavy Hitter Detection

Suneet Kumar Singh*, Christian Esteve Rothenberg*, Marcelo Caggiani Luizelli†, Gianni Antichi‡, Pedro Henrique Gomes§, Gergely Pongrácz¶
*University of Campinas, Brazil**, *Federal University of Pampa, Brazil†*, *Queen Mary University of London, UK‡*
Ericsson Research, Brazil§ and Ericsson Research, Hungary¶
*Email: *ssingh@dca.fee.unicamp.br, *chesteve@dca.fee.unicamp.br, †marceloluizelli@unipampa.edu.br*
‡g.antichi@qmul.ac.uk, §pedro.henrique.gomes@ericsson.com, ¶gergely.pongracz@ericsson.com

***Abstract*—The research community has recently proposed several solutions based on modern programmable switches to detect entirely in the data plane the flows exceeding predetermined thra eshold in a time window, i.e., Heavy Hitters (HH). This is commonly achieved by dividing the network stream into fixed time slots and identifying each separately without considering the traffic trends from previous intervals. In this work, we show that using specified time windows can lead to high inaccuracies. We make a case for rethinking how switches analyze the incoming packets and propose to leverage per-flow Inter Packet Gap (IPG) analytics instead of using flow counters for HH detection. We propose an algorithm and present a P4 pipeline design using this new metric in mind. We implement our solution on P4 hardware and experimentally evaluate it against real traffic traces. We show that our results are more accurate than related work by up to 20% while reducing the control channel overhead by up to two orders of magnitude. Finally, we showcase a QoS-oriented application of the proposed dataplane-only IPG-based HH detection in a mobile network scenario.**

***Index Terms*—Heavy-hitters detection, programmable data plane, P4, inter packet gap, network monitoring**

1. INTRODUCTION

Many network management applications benefit from identifying the flows contributing the most to the traffic, namely the so-called elephant flows or Heavy Hitters (HH), i.e. flows of size larger than or equal to a certain number of packets or bytes over a fixed time interval. This is the case, for example, for accounting [1], network capacity planning [2], load balancing [3], caching [4], or anomaly detection [5]. The recent uptake of programmable switches [6] has given the means to move the detection process directly in the data plane, enabling use cases requiring decisions at time scales comparable to traffic variations [7], such as dynamic routing [8] or flow scheduling [3]. The research community has lately worked hard towards the design and development of new algorithms that can efficiently detect heavy flows within the constraints of programmable devices [9], [10], [11], [12], [13], [14].

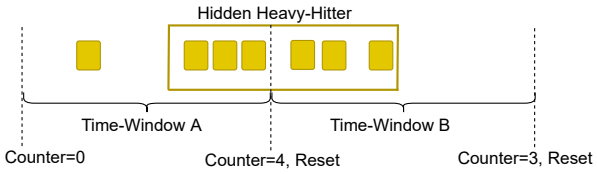


Figure 1: Counting packets over disjoint windows.

Most state-of-the-art proposals divide the time in disjoint windows and identify the HHs at the end of each of them by looking at the flows that consume more than a fraction of the link capacity. This is done with an iterative process at fixed time scales: flow counters or specific data structures, e.g., sketches, are first exported to a central controller and then reset in the data plane to deal with the new traffic [9], [10], [11], [12]. The intrinsic problem with this approach is that the final result is tightly coupled with traffic patterns and window characteristics. Figure 1 exemplifies this concept. Here, the *analyzed flow* in yellow would not be considered *heavy* because neither time window A nor B has enough occurrences. To overcome this, recent proposals have explored more complex traffic analysis mechanisms in the data plane by employing an approximation of a sliding window [15] or prefix trees that quickly adapt to traffic patterns [14]. However, simultaneous memory access is required to update a packet that enters and departs from the sliding window. Also, prefix trees need to access on-chip registers by the number of times. Both approaches are difficult to implement because of the constraints of accessing memory in programmable switch hardware [13].

Instead of counting packets belonging to each flow, we propose to use Inter Packet Gap (IPG) as the main metric to detect HH. The idea comes from the observation that the lower the IPG, the higher the flow throughput, the smaller the flow's IPG, the more packets in-flight, and the heavier the flow [16]. Notably, this simple philosophy cannot be naively applied to counters, i.e., the bigger the flow counter, the heavier the flow. This is because a flow is commonly defined as *heavy* only for a *given* time. While we would lose the time reference by adopting only counters, with IPG, this is naturally considered.

In this article, we show how failing to detect some

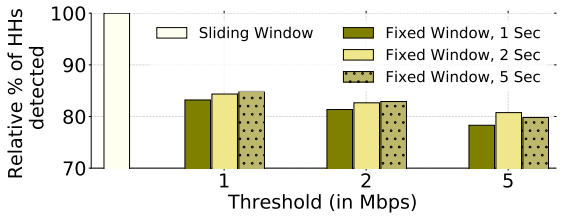


Figure 2: Analysis the number of HH flows due to the sliding and fixed window approach using CAIDA traces.

HHs can heavily impact the behavior of network-control applications such as load balancing and review the essential properties to be guaranteed by a HH detection algorithm (Sec. 2). We then discuss the use of IPG as the main metric instead of counting packets (Sec. 3) and describe how the per-flow IPG calculation can be implemented on off-the-shelf P4 programmable hardware, highlighting the challenges associated with its design and showing that existing algorithms can be adapted to support this novel metric (Sec. 4). We carry an extensive evaluation comparing state-of-the-art algorithms using real traffic traces. We show that our solution is more accurate and reduces the control channel overhead by up to two orders of magnitude (Sec. 5). In summary, the main contributions of this article are:

- We present in detail the methods for HH characterization using IPG as the key metric (first introduced in [16]) and demonstrate its practicality.
- We discuss the challenges in calculating per-flow IPG in current off-the-shelf programmable switches and implement the proposed method on P4 Tofino hardware.
- We propose an algorithm inspired by a state-of-the-art HH detection algorithm to work with IPG and experimentally evaluate its detection capabilities using real traffic traces in both simulation and P4 hardware.
- We showcase IPG-based HH detection for a QoS application in a mobile network user plane pipeline.
- We release all relevant documentation and code in an open-source repository[1] for reproducibility purposes.

2. MOTIVATION

To meet the strict requirements imposed by current programmable switches, most of the state-of-the-art proposals divide the time in disjoint windows and export data plane statistics to be used to detect HH, e.g., raw counters, sketches, at the end of each of them [9], [10], [11], [12].

Such architectural approaches suffer from the problem discussed in Figure 1. To illustrate this, we performed a number of tests using real traffic traces from CAIDA. Specifically, we used a 60-second long trace [17] and split it into 1, 2, and 5 seconds intervals (our time window for detecting heavy hitter flows). We evaluated the number of HH flows using different thresholds (i.e., 1, 2 and 5 Mbps) and fixed window sizes (i.e., 1, 2 and 5 Sec). We then did the same by using the sliding window approach to detect HH

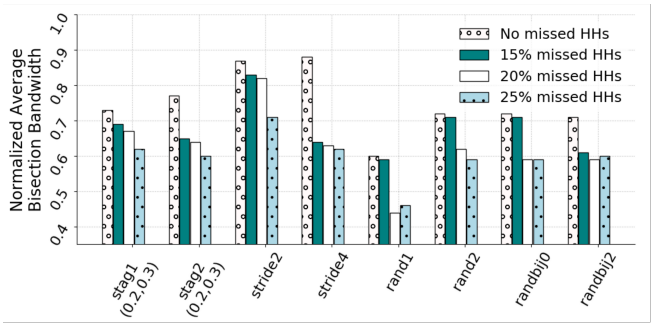


Figure 3: Impact analysis of missing HHs for load-balancing in Data Center Networks using Hedera Algorithm [3], for different communication patterns.

flows, where we measured the HH flows on every incoming packet. In Figure 2, we compare how much the true HHs detected by both fixed and sliding window approaches: the results show that the fixed window can miss the HH flows by around 15% to 20%.

To better understand how much this variability can impact a network application, we tested Hedera, a state-of-the-art intra-datacenter load balancing solution [3]. Hedera is to opportunistically schedule HH on different paths and let ECMP deal just with the short flows. Starting from the open-source code[2], we implemented a fat-tree topology (k = 4) and generated traffic following many different communication patterns, the same used in the Hedera paper. Figure 3 shows the impact on bisection bandwidth when different percentages of HH flows (i.e., 15, 20 and 25% of total HH flows) are not detected. Most importantly, we can see a performance degradation up to approximately 30% when only 15% of HH are not correctly detected.

All the problems mentioned earlier could be solved by adopting sliding-windows-based algorithms in the data plane. However, those algorithms are too complex to be implemented in off-the-shelf programmable switches [18], [19] or successfully applied when considering micro-second level timescales events, i.e., microbursts [15].

Key contribution. This work demonstrates that there is no need to completely rethink existing data plane algorithms to solve the mentioned issue. Instead, the most critical aspect to consider is the primitive adopted for HH detection. While previous works have focused on *counting packets* and optimized *data structures* to store the gathered data in the switch, we argue that IPG is a better metric for the HH detection problem. We show that it is possible to rethink state-of-the-art algorithms with this new metric and demonstrate that this can be done while meeting two fundamental properties discussed below.

Property 1: Low control channel overhead. State-of-the-art solutions leveraging data plane programmability for HH detection are based on a simple concept: the switch collects traffic statistics, stores them in appropriate data structures,

1. https://github.com/intrig-unicamp/P4-HH

2. https://bitbucket.org/msharif/hedera/

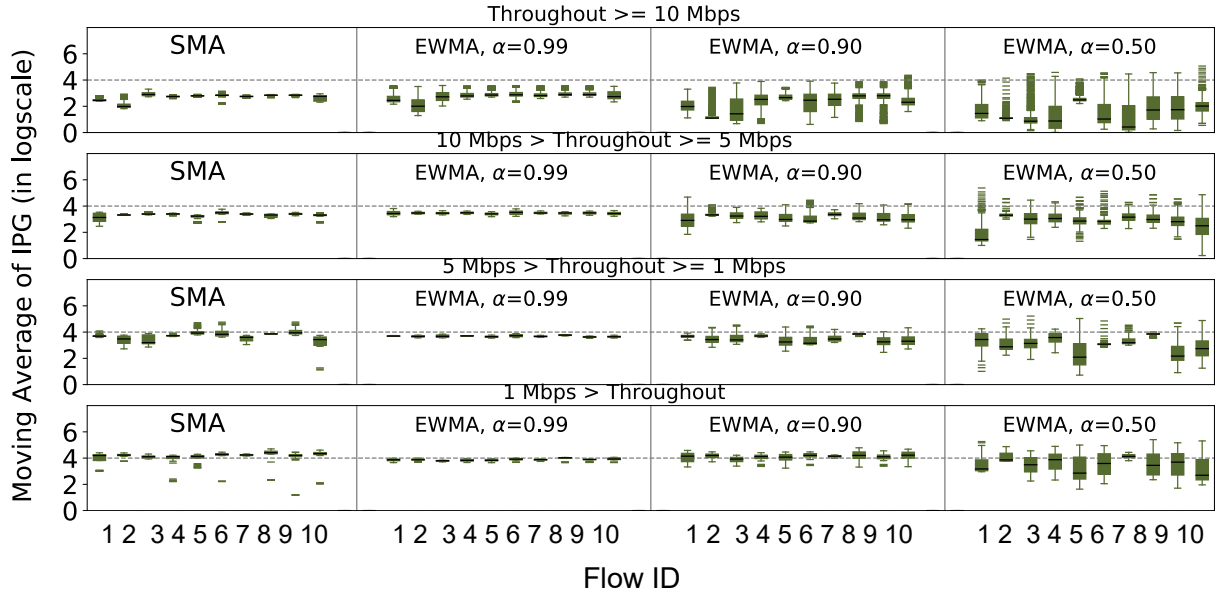


Figure 4: Distribution of EWMA ($\alpha = 0.99$ or 0.50) and SMA of IPGs of 10 flows in different throughput scale.

and exposes them regularly to a (logically) centralized controller [9], [10], [11] in charge of the detection process. This behavior can impose unnecessary load on the control plane, especially when multiple switches report their information to the same controller. Recent works have tried to mitigate this problem by proposing a push-based approach [14]. The switch exposes a single entry upon detecting a HH instead of letting the controller poll the entire data structure stored in the data plane. However, while successfully implemented on FPGA, the proposed solution appears to fail short in meeting the tight requirements in the number and the pattern of access to on-chip registers to be considered a viable solution for today's programmable switches [14].

Property 2: Low switch memory overhead. The previous properties cannot come at the cost of high memory requirements in the data plane. This is because operators need the limited and precious resources available in current programmable switches for essential control functions such as ACL rules, customized forwarding, and other network functions and applications [20].

3. IPG ANALYSIS FOR HH DETECTION

This section examines the IPG values of network flows using real-time traces and shows how they can be leveraged to detect HH flows. We consider an exponential weighting moving average (EWMA) function of the observed IPG values and optimize the degree of weighting decrease to calculate the moving average for improved accuracy.

3.1. Weighting Inter Packet Gap

IPG directly relates to the flow rate (i.e., packet size / IPG). However, as per the standard HH definition (i.e., flows size larger than or equal to the number of packets or x kB in a fixed time interval), we cannot rely only on the current IPG value. For the solution, we consider an EWMA function of the observed IPG values. In EWMA, the weighting can be set higher to decrease older observations slower. Also, the higher weighting smoothes out the sudden spikes in the IPG.

We consider a given set of network flows $f \in \mathcal{F} = \{f_1, f_2, ..., f_M\}$, where M represents the total number of flows. The IPG metric is updated every time a network packet of $f \in \mathcal{F}$ gets into the switch pipeline. When a packet enters the switch, the last seen ingress timestamp ($TS_f^l \in \mathbb{N}^+$) is subtracted from the current timestamp ($TS_c \in \mathbb{N}^+$) to calculate the current IPG_f^c estimator of network flow f (Equation 1).

$$IPG_f^c = TS_c - TS_f^l, \qquad f \in \mathcal{F} \tag{1}$$

Next, an EWMA metric IPG_f^w is computed:

$$IPG_f^w = \alpha \cdot IPG_f^{w-1} + (1-\alpha) \cdot IPG_f^c, \quad f \in \mathcal{F} \tag{2}$$

where $\alpha \in [0, 1]$ is the degree of weighting decrease and IPG_f^{w-1} is the last noted weighted IPG. IPG_f^{w-1} is initialized by IPG_{init} (i.e., $\frac{PacketSize}{HH_{th}}$), where HH_{th} indicates the HH threshold of a flow in the given time-window (TW).

The α value directly impacts the accuracy of HH detection. To optimize the α value for HH detection, we plot a graph, as shown in Figure 4. We illustrate the distribution of EWMA and SMA (Simple Moving Average) of IPG of each incoming packet of 10 flows in different throughput scales using CAIDA trace for 1 Sec TW. In SMA, weightings are equally distributed among all the IPGs of a flow, directly

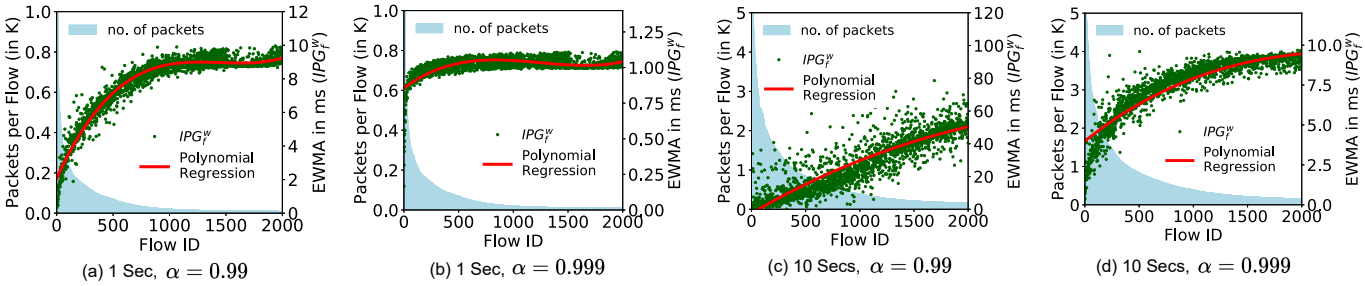


Figure 5: IPG_f^w vs Throughput. IPG_f^w indicates EWMA of IPGs at the end of TW. We consider only long-duration top-2k flows for 1 Sec TW.

related to a flow's size. We notice that EWMA at $\alpha = 0.99$ is more stable and similar to SMA than EWMA at $\alpha = 0.50$. Similar results are observed for *other real traces* such as data center (IMC10 [21]) and MAWI backbone (MAWI20 [22]) traces.

Also, we notice that for a small-duration TW (e.g., 1 Sec), where the number of packets per flow is lower than a long-duration time window, the $\alpha = 0.99$ is suitable for HH detection. However, for a long-duration time window (e.g., 10 Secs), we need to choose a higher α (e.g., around 0.999), which imposes additional challenges to calculate EWMA in the Tofino switch ASIC due to the floating-point operations required. We propose a solution to handle long-duration TW discussed in detail in Section 3.2.

The above analysis suggests that higher α values improve accuracy. We can analyze the trace and align the best possible α value to detect HH.

Weighted IPG and flow throughput. We analyze the relationship between IPG_f^w and flow throughput (total number of packets per time window (TW)). We make two cases: (1) consider only *long-duration* flows (Figures 5), (2) examine both *small* and *long*-duration flows (Figures 6). We use CAIDA traffic traces [17] and plot the number of packets per flow against the IPG_f^w calculated at the end of 1 sec TW, i.e., about 65K flows and 0.7M packets, and 10 Secs TW, i.e., around 270K flows and 5M packets. The analysis is performed for the top-2K flows. We do not observe much difference in the results for other traces [21] [22].

Case I. Figures 5a and 5d show a strong correlation between IPG_f^w and flow throughput. If we set $\alpha = 0.999$ for 1 Sec TW, we observe a weaker response to recent IPG values (Figure 5b). In Figure 5c, where $\alpha = 0.99$, some flows IPG_f^w do not correlate well with their throughput values because of the long TW duration.

Case II. In Figures 6a and 6b, considering small-duration flows with long flows, the throughput does not correlate well with IPG_f^w. This is because the lower IPG does not guarantee a higher throughput. Specifically, the flow with only two packets back-to-back can be considered HH (i.e., false positive).

As per the above discussion, we conclude that while the initial analysis points to IPG_f^w as a candidate metric for HH detection considering long-duration flows, the main challenge is handling small-size flows of lower IPG.

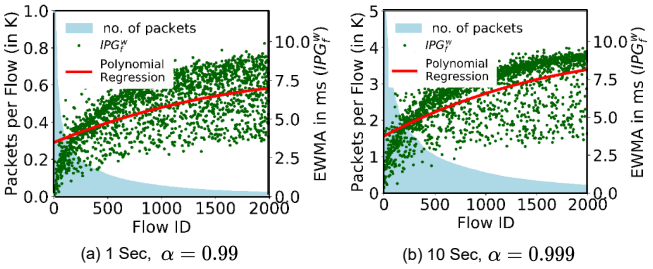


Figure 6: IPG_f^w vs Throughput, considering both small and long-duration top-2k flows for 1 Sec TW.

In the next section, we seek to answer the following questions: (1) Can we handle small-size flows, e.g., *flow only two packets back to back*, using the IPG metric? (2) How to deal with a *long-duration* time window?

3.2. Memory of Flow's Characteristic

When relying on IPG instead of packet counters to detect HH, we encounter two challenges: 1) handling *small-size flows of lower IPG*, and 2) detecting HH flows for *long-duration* TWs as discussed in subsection 3.1. To overcome these challenges, we store the throughput state of a flow based on IPG_f^w, which is used to detect HH. Maintaining IPG_f^w within the P4 switch hardware decides to keep only lower IPG_f^w flows in the data structure, while the stored throughput will decide whether the flow is HH or not.

Storing flow's throughput state. To store a flow's throughput state, we introduce a non-dimensional metric τ (i.e., compactly maintained in the programmable hardware). The τ_f metric of a flow is updated depending on IPG_f^w in a regular interval (T_{wt}). This metric is used to decide whether the flow is HH or not. We use a match-action table, where the match is performed on IPG_f^w and passes τ_f as an action parameter to store metadata and add this value in previously-stored τ_f to the switch register.

When τ_f reaches the pre-defined threshold (τ_{th}), the switch informs flow as HH to the controller for further action and reset τ_f to zero. For calculating τ_f, first, we compute the minimum number of packets (n) required for HH in a TW as follows:

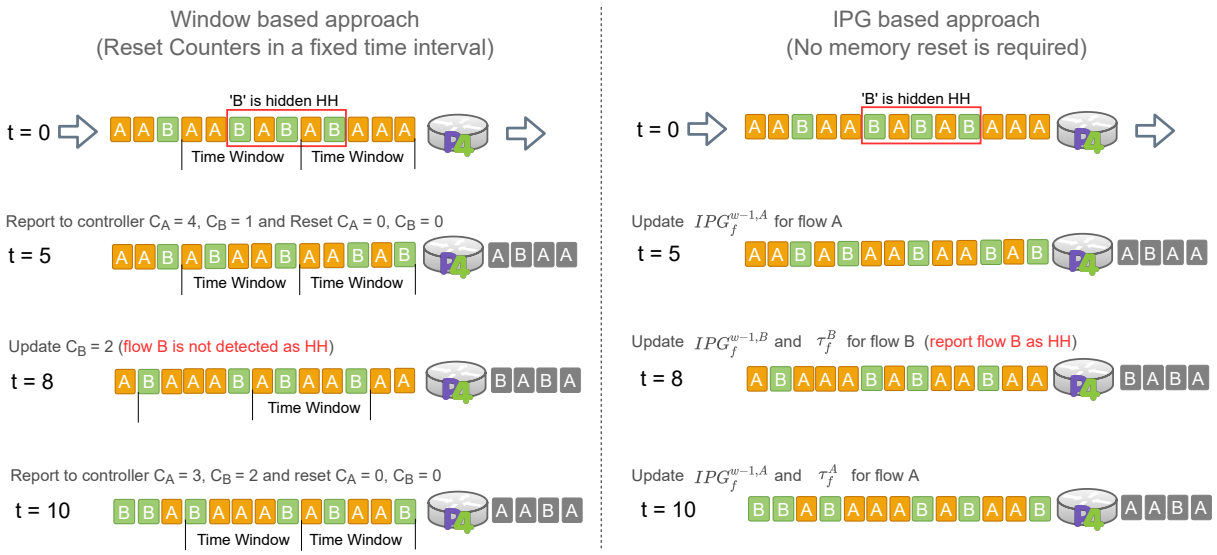


Figure 7: Example of updating metrics in both window and IPG based approaches. Considered three or more packets of a flow in a window as HH (τ_f metric is updated in every 2 secs).

$$n = \frac{TW \cdot HH_{th}}{PacketSize}, \tag{3}$$

Then, we evaluate the number of timeslots (num_{wt}) required to achieve n packets based on IPG_f^w,

$$num_{wt} = \frac{n \cdot IPG_f^w}{T_{wt}}, \tag{4}$$

where $num_{wt} \geq 2$. We divide τ_{th} by num_{wt} to get τ_f. The final expression for τ_f is as follows,

$$\tau_f = \frac{\tau_{th} \cdot T_{wt} \cdot PacketSize}{TW \cdot HH_{th} \cdot IPG_f^w}. \tag{5}$$

The controller pre-calculates the τ_f based on IPG_f^w range 1 to IPG_{init} using Equation 5 and pushes the table entries to the switch. We should choose τ_{th} to generate the maximum distinct τ_f integer values for IPG_f^w. For example, if we set TW=1 Sec, HH_{th}=10 Mbps, and T_{wt}=20 msec, we can generate a maximum of 48 distinct τ_f values for τ_{th}=1850. If we choose τ_{th} lower than 1850 (e.g., 500), 33 distinct τ_f values can be generated, impacting the accuracy and reporting time.

We follow the standard definition of HH (i.e., number of packets in a fixed TW greater than or equal to HH_{th}) used in [9] [11]. If τ_f of any flow reaches τ_{th}, this indicates that the flow's throughput is HH_{th} or above in fixed TW.

Steps to detect HH flows. Figure 7 shows an example to update the τ_f and IPG_f^{w-1} to detect HH flows. Also, the figure illustrates the difference between window and IPG based approaches. There are two flows, A and B. In the window-based approach, the packet counter of a flow is updated for each incoming packet. At t=5 (i.e., a fixed time window), the updated values of counters are informed to the controller to decide which flows are HHs and then reset both the counters to zero. Due to resetting the previous memory, we are unable to detect the flow B as HH at t=8 or t=10.

In IPG based approach, at t=5, the switch updates $IPG_f^{w-1,A}$ for flow A. Since, we do not reset any memory and update τ_f in every 2 secs, the flow B is detected as HH at t=8. The switch compares τ_f of a flow with τ_{th} and reports the flow as HH to the controller for $\tau_f \geq \tau_{th}$. τ_f for lower IPG_f^{w-1} flows increases rapidly, and the switch can report the HH flows to the controller as soon as possible. The higher IPG_f^{w-1} can quickly be evicted from the data structure to maintain only heavy flows. The detail about the algorithm is discussed in Section 4.

Correlation between τ_f and flow size . To validate the hypothesis discussed above, we analyze the correlation between τ_f or IPG_f^w and flow size using ISP (CAIDA16), data center (IMC10), and WIDE backbone (MAWI20) traces. Since the outputs from all three traces do not show a significant difference, we present the results using CAIDA in this paper. Also, analysis is performed for different throughput ranges to understand the impact of flow throughput on both IPG_f^w and τ_f metrics. We update the flow size and τ_f every 20 milliseconds, i.e., T_{wt} = 20 msec. As shown in Table 1, the τ_f highly correlates with flow size for α=0.99. This α value can easily be calculated within the Tofino HW rather than α=0.999. The details of the calculation can be found in our P4 code [23]. IPG_f^w does not correlate well with flow size because of small-size flows with lower IPGs. Also, we can observe that the correlation between τ_f and flow size decreases by decreasing flow throughput. We conclude with the following outcomes:

1) IPG_f^w and τ_f metrics allow keeping flow's throughput without any memory reset than maintaining counters, making this approach more accurate.
2) τ_f, a dimensionless metric, is used to keep flow's throughput dependent on IPG_f^w.

TABLE 1: Correlation between IPG_f^w or τ and flow size

Parameters	Flow Size (in Packets)				
	1-5 Mbps	6-10 Mbps	11-20 Mbps	20-30 Mbps	>30 Mbps
$\mathbf{IPG_f^{w,\alpha=0.50}}$	−0.13	0.063	0.22	−0.22	−0.15
$\mathbf{IPG_f^{w,\alpha=0.80}}$	−0.21	0.02	0.23	−0.23	−0.12
$\mathbf{IPG_f^{w,\alpha=0.99}}$	−0.82	−0.39	−0.03	−0.06	−0.11
$\mathbf{IPG_f^{w,\alpha=0.999}}$	−0.88	−0.75	−0.53	−0.53	−0.54
$\tau_f^{\alpha=0.50}$	0.79	0.74	0.52	0.85	0.85
$\tau_f^{\alpha=0.80}$	0.86	0.83	0.63	0.86	0.86
$\tau_f^{\alpha=0.99}$	0.91	0.95	0.96	0.97	0.96
$\tau_f^{\alpha=0.999}$	0.86	0.94	0.95	0.98	0.95

3) At α=0.99, τ_f highly correlates with flow size for HH detection to get high accuracy. Also, the same α value can be used for any duration of TW.
4) T_{wt}, a small timeslot, is used to update τ_f.
5) The small-size flows with low IPG values (e.g., flows of only two packets back-to-back) can be easily recognized as true negative using IPG_f^w and τ_f metrics.
6) The proposed method is independent of a flow duration.

4. DESIGN AND IMPLEMENTATION

4.1. P4 ASIC Implementation Challenges

P4 offers an exciting opportunity to run algorithms directly in programmable switch ASICs. However, implementing algorithms on switch ASIC brings some challenges (Figure 8). We discuss the engineering challenges illustrated in red boxes in Figure 8 and adopted solutions to implement a P4 pipeline of the proposed IPG based method validated in a programmable switch ASIC, as detailed in Section 5.

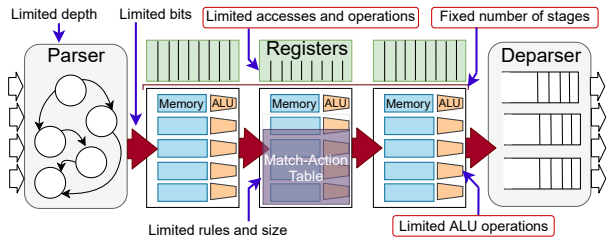


Figure 8: Resource limitations in Programmable HW.

Access Limitations to Registers. Typically, in HH detection algorithms, first, the switch checks the flow ID of incoming packets that already exist in the register. Then, the related parameters are updated, such as the packet counter. In the existing HH algorithms [12] [11], we decrease the counter and evict the flow from the data structure for unmatched Flow ID once the counter reaches zero. Since we have already visited the flow ID register to confirm the match, we cannot re-access the flow ID register to replace the entry. The register can be accessed once per packet lifetime in the Tofino switch ASIC. We face this challenge. As a solution, we rely on packet re-submission. Suppose the flow ID of incoming packets exists in the data structure (Algorithm 1: Line 7); the packets exit without any re-submission. The re-submission happens only for unmatched packets when $IPG_f^{w-1,i} > IPG_{th}$ (Algorithm 1: Line 19 to 22).

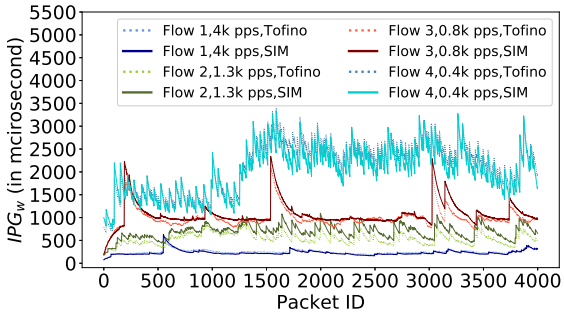


Figure 9: Running IPG_f^w for Tofino and Simulator (SIM).

The same problem discussed in [13] comes up with packet re-circulation as a solution for unmatched packets. The two main observations are noticed in [13]: (1) the expected number of recirculated packets is bounded by the square root of the number of packets which confirms the sub-linearity, (2) 1% of packets re-circulation impacted 1% on throughput. The same applies in our case to perform re-submission only for unmatched packets. The re-submission is performed efficiently within the switch ASIC pipeline and does not impact the performance much, as analyzed in Section 5.5. We use metadata to differentiate the re-submitted packet from the regular packets [23]. Also, the re-submit packet replaces the old with a new entry without any additional computation steps, as shown in Figure 10.

Arithmetic and Comparison Operations. Traditional switch HW do not support multiplications and divisions for register actions. In addition, comparison operations are limited to a fixed number of bits and can be performed between constant and variable values, not allowing the comparison of two variables. Recently available P4 programmable HW (e.g., P4 Tofino) supports bit operations that we can use to perform simple multiplications and divisions. As per Equation 2, EWMA calculation can be challenging to implement in an HW pipeline. Our solution is based on an approximate EWMA calculation that makes all the required arithmetic and comparison bit operations with limited HW resources.

Figure 9 presents the run-time IPG_f^w values in the Tofino HW and Simulator (SIM) implementations for four different flow throughput rates. In SIM, we use the standard formula to calculate the EWMA, while in Tofino, we depend on some approximations due to the P4 HW constraints. Figure 9 shows that Tofino HW and SIM do not exhibit significant differences, confirming that IPG_f^w can be used to classify HH flows directly in the Tofino HW. For future steps, we will consider a Tofino-specific extern, i.e., a *low pass filter*, that allows us to calculate EWMA, giving a more accurate calculation of EWMA of IPGs.

Fixed Number of Stages. To guarantee low and bounded per-packet latency, P4 programmable HW (Tofino) is based

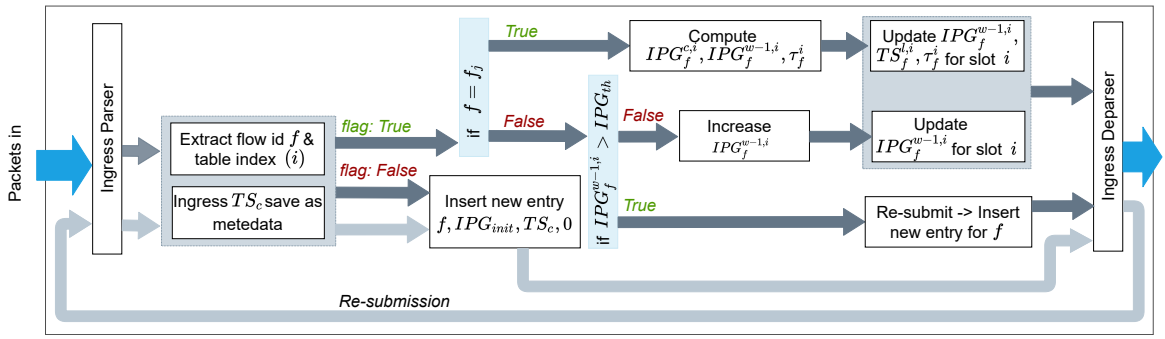


Figure 10: The proposed P4 pipeline for HH detection using novel IPG metric implemented on *Tofino switch ASIC*.

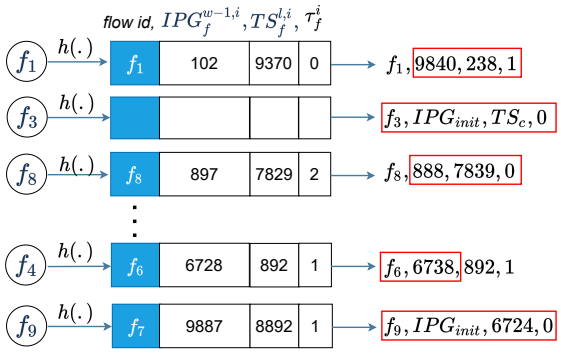


Figure 11: The data structure using IPG metric.

on a fixed number of pipeline stages. To fit our program within the available stages, we must avoid unnecessary table dependencies to perform our proposed algorithm. Currently, we need 12 stages to complete all the required steps. We run our algorithm on top of a baseline switch.p4 [24] compiled on Tofino (P4 Studio SDE 9.3.1) confirms that our algorithm can run with other switching functions without significant additional resources and stages constraints (Table 2).

4.2. Design on Programmable Switch ASIC

This section shows how to use existing data structures and algorithms with the IPG metric instead of packet count to detect HH flows accurately.

The HH decision is based on the τ_f metric following the standard HH definition discussed in Section 3.2. Let $\mathcal{P} = \{P_1, P_2, P_3, ..., P_N\}$ be a network stream with N packets. Each packet $P_j (1 \leqslant j \leqslant N)$ belongs to a given network flow $f \in \mathcal{F} = \{f_1, f_2, ..., f_M\}$, where M is the total number of network flows in the network stream $\mathcal{P}$. Each flow $f \in \mathcal{F}$ has τ_f, where $\tau_f \in \tau = \{\tau_{f_1}, \tau_{f_2}, ..., \tau_{f_M}\}$ and τ is the set of throughput state of each flow in $\mathcal{F}$. Given a network stream $\mathcal{P}$, and the pre-defined threshold τ_{th}, a HH is a flow f where $\tau_f \geq \tau_{th}$.

Our proposed method can fit most of the existing counter-based algorithms; we leverage the HeavyKeeper (HK) [12] algorithm for our prototype implementation on a Tofino hardware (HW) switch, which is amenable to programmable HW. We also analyzed our proposed approach with Space Saving [25] and Elastic Sketch [11] using a simulator. We achieved high accuracy, but due to space limitations, we focus on HeavyKeeper in this paper. We use HK-IPG to indicate the HK algorithm using IPG and HK-Count to show the HK algorithm using counters.

Data Structure. The HK-Count algorithm tries to keep only the heavy flows in the hash table, and smaller flow sizes are removed using an exponential-weakening decay scheme. When a packet enters, the main idea is if the flow ID is not the same as in the hashed table entry, HK decays the flow size by 1. When the flow size deteriorates to zero, the entry is replaced by a new flow with counter value 1.

We replace the counter with a weighted IPG (i.e., IPG_f^{w-1}) and apply the same scheme in reverse order. Instead of decaying the flow size, we increase the weighted IPG when the hash collisions occur. When the weighted IPG is higher than the pre-defined threshold, it is removed, and the new entry is inserted. We choose IPG_{th} as a threshold for evicting entries from the data structure. IPG_{th} is set the same as IPG_{init}, discussed in Section 3.1.

Insertion. As shown in Figure 11, the data structure consists of a table (i.e., T = 1,2,...,m) associated with hash function h(.) that contains m slots. Each slot includes four fields: flow id, $IPG_f^{w-1,i}$ (last noted weighted IPG), $TS_f^{l,i}$ (last noted timestamp), and τ_f^i, where $i \in T$. For this example, we consider IPG_{th} = 10000 μsec. HK-IPG checks the condition $TS_c < TS_f^{l,i}$ for each incoming packet to confirm the end of T_{wt} (timestamp wraps around in every T_{wt} [23]). If it is true, τ_f^i is updated based on $IPG_f^{w-1,i}$ and $IPG_f^{c,i}$. If $\tau_f \geq \tau_{th}$ is true, switch reports the flow as HH to the controller. Specifically, there are three cases:

Case I: The slot is empty. When the packet of f_3 enters the switch, we insert $f_3, IPG_{init}, TS_c, 0$, where we initialize the $IPG_f^{w-1,i}$ with IPG_{init} (i.e., equal to IPG_{th}), TS_c is the current time-stamp, and τ_f^i is initialized with 0.

Case II: $f = f_i$. This condition occurs for f_1 and f_8. In this case, HK-IPG calculates the IPG and updates each field of the slot accordingly. In the case of f_1, τ_f^i is updated with one based on the $IPG_f^{w-1,i}$; however, for f_8, since the condition $TS_c < TS_f^{l,i}$ is not true, τ_f^i remains the same.

Case III: $f \neq f_i$. HK-IPG checks the condition

$IPG_f^{w-1,i} \leqslant IPG_{th}$. If true, $IPG_f^{w-1,i}$ is increased linearly by adding k, which is pre-defined based on α and table size. In this example, we consider k=10. When the condition $IPG_f^{w-1,i} > IPG_{th}$ becomes true, the existing entry is replaced by the new entry. For f_4, ten is added in the field $IPG_f^{w-1,i}$. In the case of f_9, after updating the field $IPG_f^{w-1,i}$, the condition $IPG_f^{w-1,i} > IPG_{th}$ is true. HK-IPG replaces the entry with the new entry.

Figure 10 depicts the high-level view of the proposed P4 pipeline to detect HH flows. Algorithm 1 represents the detailed description of the HK-IPG algorithm and explains the three main steps for keeping the per-flow state.

4.3. Limitations and Corner Cases

Limitations. Currently, our solution requires at least two timeslots ($num_{wt} \geq 2$) discussed in Section 3.2 to detect HH flows, which makes undetectable short-lived HH flows that appear in a single timeslot. We can reduce the probability of missing these flows by reducing the length of the timeslot. However, in that case, we may lose the overall accuracy in detecting HH flows.

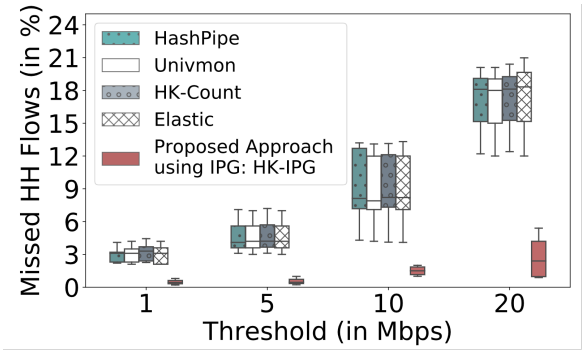
Moreover, our approach may not detect HH flows with a specific pattern, where the inter-burst gap is more significant than the timeslot (T_{wt}) due to failure to check the condition: $TS_c < TS_f^l$. Thus, although we optimize the size of T_{wt} to lower the possibility of missing this type of HH flow, it is still possible to miss a few HHs. However, we can completely diminish this limitation with some memory expenses by keeping a flow's starting time-stamp and updating that at the end of each T_{wt}.

Corner Cases. Using the IPG metric for HH detection, there could be a case where a flow only contains two packets back to back (i.e., low IPG). *Do we treat such flows as HH using our proposed approach ?*. The answer is no. As we discussed in Section 3.2, to resolve this issue, we use multiple timeslots to decide whether the flow is HH or not. Also, we initialize IPG_f^{w-1} with the same value as IPG_{th}. If the flow has a lower IPG with only a few packets, there is a small probability of reducing IPG_f^{w-1} significantly. It is also unlikely that these types of flows survive in multiple timeslots with low IPG.

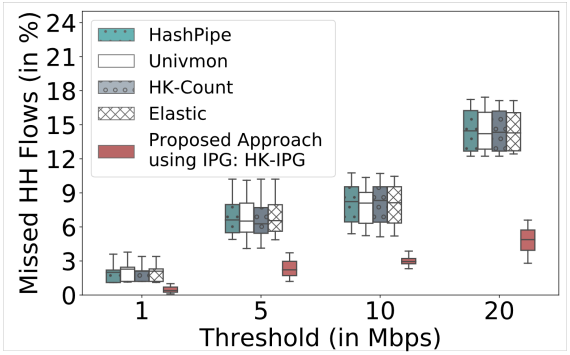
Since we choose a small timeslot (T_{wt}) to update the τ_f as discussed in Section 3.2, there could be another corner case: *is the proposed approach limited detecting the HH flows for small length TW only?* Note that we use T_{wt} to update the τ_f metric for keeping the throughput state of a flow. There is no direct relationship between the timeslot and the length of the time window. The detailed discussion can be found in Section 3.2.

5. EXPERIMENTAL RESULTS

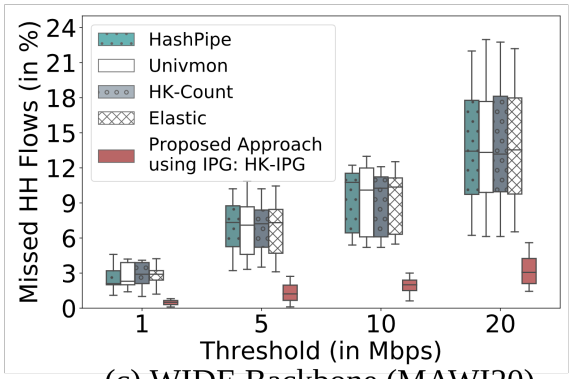
We analyze the performance of HK-IPG using the Tofino HW switch implementation with real traffic traces and compare it with the state-of-the-art. We detect HHs using different flow ID definitions such as 5-Tuple, source IP, and destination IP. Furthermore, four different measuring time intervals are investigated: 1, 5, 10, and 60 Secs.



(a) ISP backbone (CAIDA16)



(b) Data Center (IMC10)

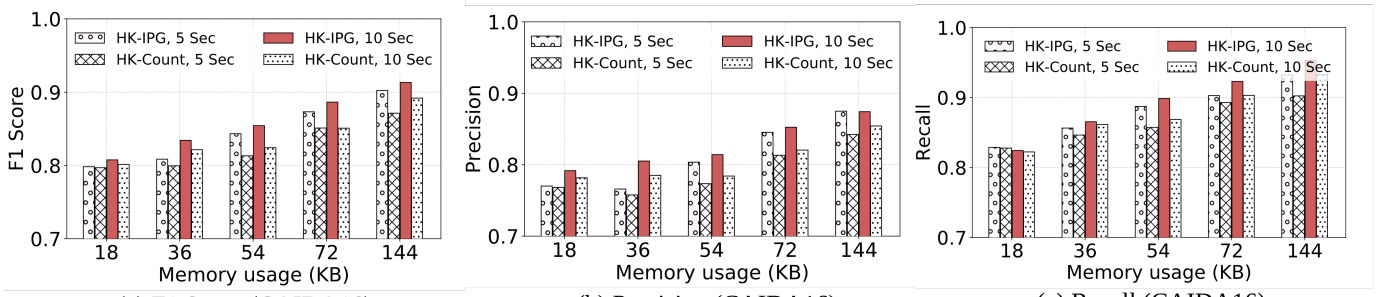


(c) WIDE Backbone (MAWI20)

Figure 12: Comparison of the proposed algorithm with the state-of-the-arts. Figures (a), (b) and (c) show the missed HH flows due to memory resets.

5.1. Experiment Setup

Testbed. We evaluate the performance of our proposed IPG based HH detection method on a *Barefoot Tofino switch ASIC* (Edgecore Wedge 100BF-32X). We use the FPGA-based OSNT [26] and the DPDK-based TRex [27] as the traffic generators on an x86 server (Intel Xeon D-1518) with 10G SFP+ interfaces connected to the Tofino switch under



(a) F1 Score (CAIDA16) (b) Precision (CAIDA16) (c) Recall (CAIDA16)

Figure 13: Figures (a), (b) and (c) illustrate the overall accuracy and compare with HK-Count.

Algorithm 1: Proposed Algorithm: HK-IPG

Input: A packet $P_j (1 \leqslant j \leqslant N)$ belonging to flow ID f.
Get table index i from $h(f)$, where $i \in T = (1, 2, .., m)$.
if *flag* = *false* **then**
  /* Case 1 */
  $flag \leftarrow true$;
  $f_i, IPG_f^{w-1,i}, TS_f^{l,i}, \tau_f^i = f, IPG_{init}, TS_c, 0$;
else
  if $f = f_i$ **then**
    /* Case 2 */
    if $TS_c > TS_f^{l,i}$ **then**
      $IPG_f^c = TS_c - TS_f^{l,i}$;
      $IPG_f^{w-1,i} = \alpha \cdot IPG_f^{w-1,i} + (1-\alpha) \cdot IPG_f^c$;
      $TS_f^{l,i} = TS_c$;
    else
      $IPG_f^c = TS_c + T_{wt} - TS_f^{l,i}$;
      $IPG_f^{w-1,i} = \alpha \cdot IPG_f^{w-1,i} + (1-\alpha) \cdot IPG_f^c$;
      $TS_f^{l,i} = TS_c$;
      match on $IPG_f^{w-1,i}$, set $metadata.tau$ as an action;
      τ_f^i= τ_f^i + $metadata.tau$;
  else
    /* Case 3 */
    if $IPG_f^{w-1,i} \leqslant IPG_{th}$ **then**
      $IPG_f^{w-1,i} = IPG_f^{w-1,i} + k$;
    else
      $(f, IPG_f^{w-1,i}, TS_f^{l,i}, \tau_f^i =$
       $f, IPG_{init}, TS_c, 0)$;

test (DUT). Our proposed design, implemented in P4, is executed on the Tofino switch. The traffic generators replay the real time traffic traces to the Tofino switch. The running algorithm detects the HH flows on run-time and informs the controller for further processing.

Dataset. We use the CAIDA-2016 [17] ISP backbone link, IMC-10 [21] data center, and MAWI-20 [22] WIDE backbone traces for evaluation.

CAIDA16. The traffic trace is around 60 Secs and contains 20 Million packets and 800K flows. We split this trace into 1, 5, and 10 Secs small chunks. The 5 Secs chunk contains around 2 Million packets, and 150K flows, while the 10 Secs chunk carries about 4.5 Million packets and 280K flows.

MAWI20. The WIDE backbone traces are 15 minutes long with 600K packets (4000 flows) per second.

IMC10. The data center traces are 20 minutes long with approximately 15K packets (200 flows) per second. We replay the packets at a higher rate (i.e., about 3500 flows per second), following the exact state-of-the-art [10].

Implementation. For the programmable switch ASIC, the P4 pipelines of HK-IPG (i.e., around 500 lines of code [23]) algorithm is implemented using the Tofino Native Architecture (TNA). A single Table is maintained with m number of memory slots. Each slot contains four values: 32-bit flow ID, 16-bit IPG_f^{w-1}, 16-bit TS_f^l, and 8-bit τ_f. Total memory use is calculated by $m \times 72(bits)$. For setting the weighted IPG threshold, we assume a default packet size of 1000 Bytes. Also, the value of α is set to 0.99 for the EWMA calculation. Flows above 1 Mbps are considered as HHs. We get around the same results for other higher HH threshold values.

5.2. Evaluation Metrics

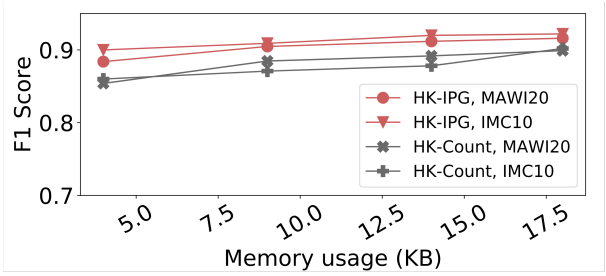
F1 Score, Recall and Precision. We define as TP (True Positive) those HH flows which are correctly detected, FP (False Positive) those flows which are non-HH but detected as HH, and FN (False Negative) the flows which are HH but detected as non-HH. We evaluate the Precision ($Pr = \frac{TP}{TP+FP}$) and Recall ($R = \frac{TP}{TP+FN}$) metrics, from which we derive the F1 score as $\frac{2 \cdot (R \cdot Pr)}{(R+Pr)}$.

Bandwidth. We measure the control channel overhead in bandwidth (kB) per sec for different reporting time intervals.

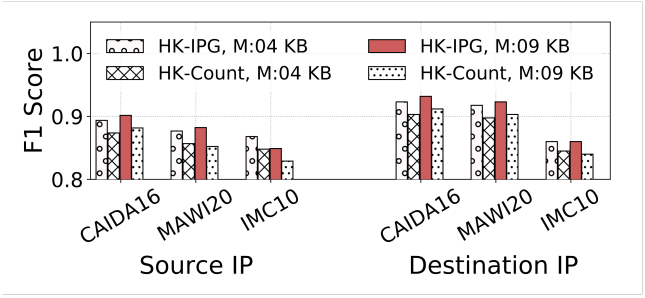
Throughput and Latency. We measure the computational complexity of our proposed algorithm in terms of throughput and latency on the Tofino switch with and without the proposed algorithm. The throughput defined in packets per second is measured with TRex, while latency is measured one-way end-to-end using OSNT.

5.3. Accuracy

As discussed in Sections 1 & 2, in most of the existing disjoint time window-based algorithms, we can miss the true HH flows because of resetting the previous interval memory, as shown in Figure 1. We compare our proposed algorithm



(a) 5-Tuple flow definition



(b) other flow definitions

Figure 14: Accuracy with IMC10 and MAWI20 traces.

with the state-of-arts. We perform the Tofino HW analysis to compare our solution with HK-Counter. To compare with other existing algorithms, we perform the simulator analysis since it is challenging to implement all the other state-of-arts on Tofino switch ASIC and some of the P4 source code's unavailability. We use the 60 Secs long *CAIDA 2016* trace [17] to detect the true HH flows for 1 Sec TW by applying the *sliding window* approach. The complete simulation and P4 code are available on GitHub [23]. To find the true missed HH flows due to resetting the counters in every 1 Sec TW, we split CAIDA traces into 1 Sec TWs and evaluate the true HH flows. Second, we apply state-of-art algorithms to detect the number of missed HH flows out of true HH flows due to reset counters. In the same way, we use our algorithm to detect the number of missed HH flows. The same steps are performed for *IMC10* and *MAWI20* traces.

Figures 12 (a), (b) and (c) show that our proposed algorithm can detect most of the missed HH flows. Only 0.2-5.0 % of true HH flows are not detected by HK-IPG, while up to 20% of true HH flows are not exposed in the case of state-of-art algorithms.

Impact of missing HH flows. As discussed in Section 2, if we miss 15-25% of HH flows, around 30% of bisection bandwidth can be impacted for load balancing in data center networks. Also, we analyze the impact of HH detection on other real-time applications such as QoS for the mobile network, discussed in detail in Section 6.

We assess the impact of memory and the duration of measuring time-interval on precision and recall. Figures 13 (a), (b), and (c) present the results of the HK-IPG and HK-Count algorithms on the ***Tofino switch ASIC***. We observe that our proposed solution can detect HHs with high accuracy compared with the HK-Count approach. We analyze

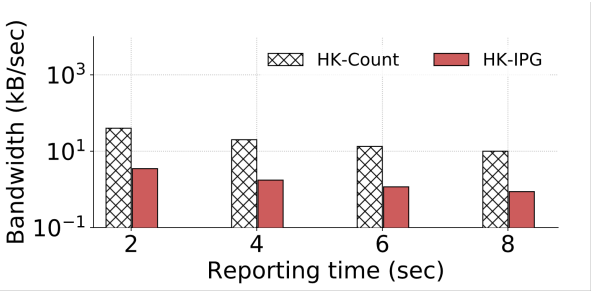


Figure 15: Bandwidth Utilization

the performance for different measuring time intervals (i.e., 5 and 10 Secs) and find that precision and recall values are consistent for larger memory sizes. For smaller memory sizes (i.e., 18 KB), the probability of evicting the HH flows from the data structure increases.

For larger memory sizes, the F1 Score value is above 0.90, confirming that precision and recall are balanced enough to provide high accuracy. Also, we evaluate our proposed algorithm with different flow definitions and other real traces, IMC10 [21] and MAWI20 [22], as shown in Figure 14. We achieve high accuracy for both data center and WIDE backbone network traces compared with the existing counter based approach.

5.4. Bandwidth Utilization

Figure 15 compares the amount of data exchanged between a switch and a controller. We analyze the performance using CAIDA traces by setting 1 Mbps as the threshold for different reporting times. The switch reports the HH flows to the controller when they turn into the HH. It is confirmed from Figure 15, our proposed algorithm can save a significant amount of control plane traffic, around two orders of magnitude compared to the state-of-the-art.

5.5. Computational Complexity

To examine the computational complexity of the proposed approach, we perform throughput (TP) and latency (LAT) tests using CAIDA traces and compare them with state-of-art algorithms. Since only a few existing algorithms are implemented on P4 switch ASIC, and there is a lack of availability of TNA (Tofino Native Architecture) P4 codes, we choose HK-Count for comparison. First, we measure TP and LAT without using any algorithm (i.e., just output packet forwarding without applying any actions). In the second case, we measure the performance with the HK-IPG and HK-Count algorithms. As we can observe in Figure 16, the throughput is not affected in both cases, as expected from a line-speed hardware pipeline. The difference in the latency with and without the HK-IPG algorithm is 176 nanoseconds.

5.6. Resource Utilization

We observe no significant difference in maintaining per user state in the P4 switch ASIC compared with existing

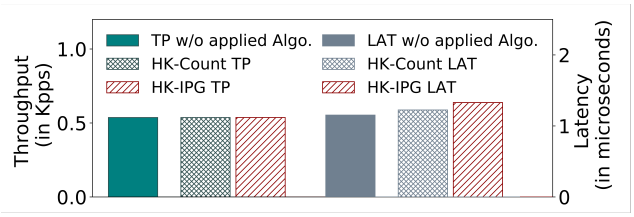


Figure 16: Performance on *Tofino Switch ASIC*.

TABLE 2: Additional HW *Tofino resources* used by HK-IPG, running on top of a baseline *switch.p4*.

Resource	Baseline (%)	HK-IPG with Baseline(%)	Additional usage(%)
Exact Match Input Crossbar	16.9	21.6	4.7
Hash Bit	20.1	28.5	8.4
Hash Distribution Unit	16.7	45.8	29.1
Meter ALU	12.5	22.9	10.4
SRAM	23.6	25.1	1.5
TCAM	21.9	22.2	0.3
VLIW Instruction	12.8	16.7	3.9
Exact Match Result Bus	18.8	27.6	8.8

algorithms. As discussed in Section 5.1, for maintaining per user state in P4 switch ASIC, 72 bits are required for our proposed approach, while other existing algorithms generally need 32 bits for flow Id and 32 bits for counters. We run our algorithm on top of a baseline switch.p4 [24] (P4 Studio SDE 9.3.1) compiled on Tofino switch ASIC. In Table 2 (additional usage), we can see that all resources use around less than 10%, except for the hash distribution unit. The hash distribution unit is used to map hash output to Packet Header Vector containers without using any table lookup. Stateful ALU is used in each stage to read and write operations in the register array. SRAM is used for storing the metrics of the HK-IPG algorithm.

6. Use Cases

QoS-HH. After HH detection, different traffic management and control actions are typically carried at different time scales, e.g., DCTCP [28], DDoS attack [29], IntSight [30], H-UPF [31]. To analyze the proposed approach, we evaluate the Flow Completion Time of the pipeline configured to send all IPG detected HHs to lower priority queues. At the same time, delay-critical flows go to a higher priority queue.

As shown in Figure 17(a), we consider a use case scenario of a mobile network implemented by adding our `HK-IPG.p4` functional blocks to a mobile gateway P4 pipeline implementation (`vEPG.p4`) [32]. The P4 code is compiled in a Tofino hardware connected to three different hosts acting as eNodeB, Application Server, and Upstream Router facing the Internet.

We use Stratum [33] to configure the port shaping rate from 10G to 1G interface to create a congestion environment. The required entries, such as match-action parameters (i.e., IPG_f^w and τ_f) discussed in Section 3.2 and other forwarding entries, are pushed using P4 Runtime [34]. 20 different TCP flows sharing the available link bandwidth, considered delay critical, are taken to download the 15 Mbytes data from the Internet. We evaluate FCT for each delay critical TCP flow.

We consider three different test-case scenarios. First, as a traditional setup, we observe the FCT of delay-critical flows with concurrent HH flows (occupying 40% of link bandwidth with around 80 Mbps flow rate). In the second and third cases, we demonstrate the FCT gains when HH are detected and sent to lower priority queues using HK-IPG and HK-Count approaches. Since the higher priority queues are served as a strict priority over lower priority queues in Tofino, we can see the difference in FCT for the different scenarios (Figure 17(b)). The FCT can be improved by around 16-24% using our proposed IPG based approach.

Other Applications. The proposed approach can also be used for other application tasks. We briefly sketch two applications leveraging the IPG metrics.

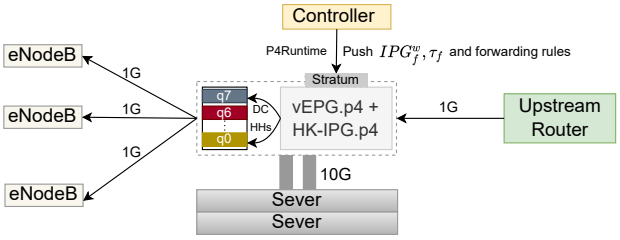
Microbursts detection. Short-lived traffic surges are known as microbursts. In the data-center network, microbursts can be the cause of packet loss due to fully utilized queues. In the recent work [7], microburst culprits are detected in programmable switch ASIC using a sliding window approach. However, the existing approach supports a single queue to detect microburst flows. In reality, there are multiple queues in a switch, which turns out to be very challenging to detect microburst flows within multiple queues at a time. In our case, there is no need to rethink the proposed HK-IPG algorithm, which can be directly used to detect HH flows on a microsecond scale (around 92% accuracy in our initial evaluation) with multiple queue support.

Network-wide telemetry. Carrying per switch IPG metric in packet headers through In-band Telemetry (INT) [35] could open new avenues to identify network performance issues and provide novel visibility means for root cause analysis through machine learning methods applied to network-wide per-hop IPG metrics over time. The intuition being explored [36] is that IPG variations over time across different flows and network paths can be valuable insights to track performance issues throughout the network, eventually back to the source.

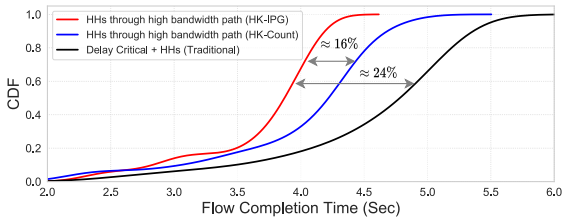
7. RELATED WORK

In this section, we discuss in detail the related works. IPG analysis has been employed in some networking applications [37] [38] [39], but never to the HH detection problem to the best of our knowledge. In the past, the detection of heavy flows was performed outside the data plane in software collectors. Network devices employ packet sampling and exported statistics using well-known protocols such as NetFlow [40] or sFlow [41] to lower overheads and data collection bandwidth at the cost of estimation accuracy.

Recently, the uptake of programmable data planes has enabled the possibility of involving switches in the traffic analysis process. Specifically, several efforts have been proposed to count every packet belonging to every flow and store this information in probabilistic data structures,



(a) QoS-HH Use-case Scenario



(b) Flow completion time with and without HHs offloading to lower priority queue

Figure 17: Mobile network scenario (a) and distribution of Flow Completion Times (FCT) (b) of delay critical flows showcasing the gains of sending HHs detected by HK-IPG (red) to lower priority queues.

i.e., sketches [9], [11], [42], [43], to be implemented in the switch ASIC and exported at regular intervals to a central controller for further processing analysis. Some other proposals have suggested counting only on the heavy flows [10], [12], [25] to increase the detection accuracy as much as possible. The idea is to dynamically evict from a probabilistic data structure, stored in the switch ASIC, the contribution of short flows, thus minimizing as much as possible the effect of small flows estimating of the large ones. In all cases, these counting algorithms rely on a periodic reset of the switch's memory at the end of each TW. This makes those solutions prone to the problem shown in Figure 1.

Implementing a sliding-window based algorithm would circumvent the aforementioned issue. In this context, Memento [19], WCSS [18], SWAMP [44], and [45] maintain the last n packets of a network stream to detect HHs. However, implementing those data structures considering the restriction of today's programmable data planes is not possible. Finally, other approaches decoupled from using windows by proposing a prefix tree can expand or collapse over time [14], [46]. However, in this case, due to limitations in the number of register accesses, these algorithms are not feasible for today's programmable switches.

Other efforts, such as [47] storing the ingress time of the first and last packets of a specific flow and accumulating the size of each incoming packet, have been considered to detect heavy flows in the P4 switch. To decide whether the packet belongs to an already existing flow or not, the time interval between the last and current packet is calculated and compared with the pre-defined timeout. However, optimizing the timeout value is difficult, leading to a false positive. In [48], an algorithm is proposed for incremental deployment of SDN programmable switches in legacy infrastructures to monitor network-wide heavy-hitter flows. This approach is based on the time window, wherein each time window, the programmable switches dynamically store only the heavy flows, and all the statistics are reset at the end of the time window. This strategy can miss the hidden heavy hitters discussed in Figure 1. In [49], heavy flows are detected within the switch and re-routed, colliding on the same path in multi-rooted network topology. There is multistage processing in detecting and re-routing heavy flows. Based on the flow id, the flow size is compared with the pre-defined threshold in a given time window at the time of each incoming packet. Then, reset the previous memory at the end of the fixed time window.

8. CONCLUSIONS AND FUTURE WORK

We presented a completely different approach using *Inter Packet Gap (IPG)* analytics to detect heavy hitters entirely in the data plane. Our proposed method supports a push-based approach, where the data plane reports the controller simultaneously when the flows turn into heavy hitters. Our design has been implemented within the constraints of a programmable switch ASIC. Also, it is easy to be adapted to most of the existing counter-based algorithms with high accuracy. We performed our evaluation on the Tofino switch ASIC using real traces and achieved high accuracy and low control channel overhead compared to the state-of-art algorithms. We showcase the QoS benefits of blending dataplane IPG-based HH detection with dynamic queue allocation.

Our future work includes a set of flourishing paths. We are looking into probabilistic data structures like sketches instead of simple hash tables on bit space optimization. Along that path, we are investigating how IPG-based methods could be generalized beyond HH detection applications for network-wide anomaly detection using INT for IPG metric observability [36]. The use of adequate query languages of IPG metrics (per device and network-wide after INT collections) is another investigation piece for novel applications such as root cause analysis.

Finally, regarding use cases of IPG metrics for HH rankings, we are working on improved building blocks for hybrid SW/HW P4 pipelines [31] applied to 5G user plane functions, where HH flows are kept in the P4 HW pipeline. At the same time, x86 SW handles lightweight flows, providing a scalable and cost-efficient sweet spot design for VNF HW acceleration.

Acknowledgments

This work was partially supported by the Innovation Center, Ericsson S.A., Brazil, grants UNI.66 and UNI.70. Marcelo C. Luizelli has been partially funded by São Paulo Research Foundation (FAPESP) – Grant 2021/06981-0, Brazilian Brazilian Council for Scientific and Technological Development (CNPq) – Grant 404027/2021-0, and Rio Grande do Sul Research Foundation (FAPERGS) – Grant 21/2551-0000688-9.

References

[1] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting," in *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2002.

[2] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "Deriving Traffic Demands for Operational IP Networks: Methodology and Experience," in *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2000.

[3] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *Networked Systems Design and Implementation (NSDI)*. USENIX, 2010.

[4] O. Rottenstreich and J. Tapolcai, "Optimal Rule Caching and Lossy Compression for Longest Prefix Matching," in *Transactions on Networking, Volume:25, Issue:2*. IEEE/ACM, 2017.

[5] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing Network-wide Traffic Anomalies," in *Computer Communication Review, Volume: 34, Issue: 4*. ACM, 2004.

[6] Intel (Barefoot Networks division), "Tofino chip," https://www.barefootnetworks.com/technology/.

[7] T. Benson, A. Anand, A. Akella, and M. Zhang, "MicroTE: Fine Grained Traffic Engineering for Data Centers," in *COnference on Emerging Networking EXperiments and Technologies (CoNEXT)*. ACM, 2011.

[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling Flow Management for High-performance Networks," in *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2011.

[9] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon," in *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2016.

[10] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-Hitter Detection Entirely in the Data Plane," in *Symposium on SDN Research (SOSR)*. ACM, 2017.

[11] 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 *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2018.

[12] T. Yang, H. Zhang, J. Li, J. Gong, S. Uhlig, S. Chen, and X. Li, "HeavyKeeper: An Accurate Algorithm for Finding Top-k Elephant Flows," in *Transactions on Networking, Volume:27, Issue:5*. IEEE/ACM, 2019.

[13] R. Ben Basat, X. Chen, G. Einziger, and O. Rottenstreich, "Designing Heavy-Hitter Detection Algorithms for Programmable Switches," in *Transactions on Networking, Volume:28, Issue:3*. IEEE/ACM, 2020.

[14] J. Kučera, D. A. Popescu, H. Wang, A. Moore, J. Kořenek, and G. Antichi, "Enabling Event-Triggered Data Plane Monitoring," in *Symposium on SDN Research (SOSR)*. ACM, 2020.

[15] X. Chen, S. L. Feibish, Y. Koral, J. Rexford, O. Rottenstreich, S. A. Monetti, and T.-Y. Wang, "Fine-Grained Queue Measurement in the Data Plane," in *Conference on Emerging Networking Experiments And Technologies (CoNEXT)*. ACM, 2019.

[16] S. Singh, C. E. Rothenberg, M. C. Luizelli, G. Antichi, and G. Pongracz, "Revisiting heavy-hitters: Dont count packets, compute flow inter-packet metrics in the data plane," in *SIGCOMM Demos and Posters*, 2020.

[17] CAIDA, "Anonymized Internet Traces," http://www.caida.org/data/passive/passive_dataset.xml.

[18] R. Ben Basat, G. Einziger, R. Friedman, and I. Keslassy, "Heavy Hitters in Streams and Sliding Windows," in *International Conference on Computer Communications (INFOCOM)*. IEEE, 2016.

[19] R. B. Basat, G. Einziger, I. Keslassy, A. Orda, S. Vargaftik, and E. Waisbard, "Memento: Making Sliding Windows Efficient for Heavy Hitters," in *Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*. ACM, 2018.

[20] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "NetCache: Balancing Key-Value Stores with Fast In-Network Caching," in *Symposium on Operating Systems Principles (SOSP)*. ACM, 2017.

[21] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 267280.

[22] MAWI, "Working Group Traffic Archive," https://mawi.wide.ad.jp/mawi/ditl/ditl2020-G/.

[23] The P4 Source Code of IPG based Heavy-Hitter detection for Tofino, [Available]:https://github.com/intrig-unicamp/P4-HH.

[24] Open-Source P4 Implementation of Features Typical of an Advanced l2/l3 Switch, [Available]:https://github.com/p4lang/switch.

[25] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient Computation of Frequent and Top-k Elements in Data Streams," in *International Conference on Database Theory (ICDT)*. Springer-Verlag, 2005.

[26] G. Antichi, M. Shahbaz, Y. Geng, N. Zilberman, A. Covington, M. Bruyere, N. Mckeown, N. Feamster, B. Felderman, M. Blott, A. W. Moore, and P. Owezarski, "Osnt: open source network tester," *IEEE Network*, vol. 28, no. 5, pp. 6–12, 2014.

[27] TRex, "Cisco TRex Traffic Generator ," https://trex-tgn.cisco.com/.

[28] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 6374.

[29] . C. Lapolli, J. Adilson 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)*, April 2019, pp. 19–27.

[30] J. Marques, K. Levchenko, and L. Gaspary, "Intsight: Diagnosing slo violations with in-band network telemetry," in *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 421434.

[31] S. Kumar Singh, C. E. Rothenberg, J. Langlet, A. Kassler, P. Vörös, S. Laki, and G. Pongrácz, "Hybrid p4 programmable pipelines for 5g gnodeb and user plane functions," *IEEE Transactions on Mobile Computing*, pp. 1–18, 2022.

[32] S. K. Singh, C. E. Rothenberg, G. Patra, and G. Pongracz, "Offloading virtual evolved packet gateway user plane functions to a programmable asic," in *Proceedings of the 1st ACM CoNEXT Workshop on Emerging In-Network Computing Paradigms*, ser. ENCP '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 914.

[33] Stratum, "An open source silicon-independent switch operating system for software defined networks." [Online]. Available: https://opennetworking.org/stratum/

[34] P4Rntime, "A control plane specification for controlling the data plane elements of a device or program defined by a p4 program." [Online]. Available: https://github.com/p4lang/p4runtime/

[35] L. Tan, W. Su, W. Zhang, J. Lv, Z. Zhang, J. Miao, X. Liu, and N. Li, "In-band network telemetry: A survey," *Computer Networks*, vol. 186, p. 107763, 2021.

[36] F. Germano Vogt, F. Rodriguez, C. Rothenberg, and G. Pongrácz, "Innovative network monitoring techniques through in-band inter packet gap telemetry (ipgnet)," *5th P4 Workshop in Europe 2022 (EuroP4 '22)*, 2022.

[37] F. Hao, M. Kodialam, and T. V. Lakshman, "Line-rate, real-time-traffic detector," Nov. 8 2011, uS Patent 8,054,760.

[38] Y. Sun, V. Xu, S. Ahlawat, and K. Daftary, "Measuring network performance using inter-packet gap metric," Nov. 3 2015, uS Patent 9,178,783.

[39] F. Tang, H. Zhang, L. T. Yang, and L. Chen, "Elephant flow detection and differentiated scheduling with efficient sampling and classification," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2019.

[40] Cisco, "Cisco IOS Netflow," http://cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html.

[41] "sFlow," https://sflow.org/.

[42] Q. Xiao, Z. Tang, and S. Chen, "Universal online sketch for tracking heavy hitters and estimating moments of data streams," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 974–983.

[43] Y. Fu, D. Li, S. Shen, Y. Zhang, and K. Chen, "Clustering-preserving network flow sketching," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 1309–1318.

[44] R. Ben Basat, G. Einziger, R. Friedman, and Y. Kassner, "Poster abstract: A sliding counting bloom filter," in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2017, pp. 1012–1013.

[45] B. Turkovic, J. Oostenbrink, and F. Kuipers, "Detecting Heavy Hitters in the Data-plane," in *CoRR, abs/1902.06993*. Cornell University, 2019.

[46] N. Kammenhuber and L. Kencl, "Efficient statistics gathering from tree-search methods in packet processing systems," in *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, vol. 3, 2005, pp. 1483–1489 Vol. 3.

[47] M. V. B. da Silva, A. S. Jacobs, R. J. Pfitscher, and L. Z. Granville, "Ideafix: Identifying elephant flows in p4-based ixp networks," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–6.

[48] 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.

[49] D. Sanvito, A. Marchini, I. Filippini, and A. Capone, "Cedro: an in-switch elephant flows rescheduling scheme for data-centers," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, 2020, pp. 368–376.

Suneet Kumar Singh is a year PhD student at UNICAMP in the School of Electrical and Computer Engineering, Brazil. He received M.Tech in communications engineering from Galgotias University, India, in 2014 and B.Tech in electronics & communication engineering from SITM college, India, in 2008. He worked in Open Network Foundation (ONF) as a software engineer in the Aether project from 2020-2021.

Christian Esteve Rothenberg is Head of the Information & Networking Technologies Research & Innovation Group (INTRIG) and Associate Professor at University of Campinas (UNICAMP), where he received his Ph.D. in Electrical and Computer Engineering in 2010. From 2010 to 2013, he worked as Senior Research Scientist in the areas of IP systems and networking at CPqD R&D Center in Telecommunications, Brazil. He holds the Telecommunication Engineering degree from the Technical University of Madrid, Spain, and the M.Sc. (Dipl. Ing.) degree in Electrical Engineering and Information Technology from the Technical University of Darmstadt, Germany, 2006.

Marcelo Caggiani Luizelli is an associate Professor at Federal University of Pampa (Brazil). He holds a Ph.D. degree in Computer Science (UFRGS, 2017). His interests are computer networks, algorithms, and optimization, focusing on NFV, SDN, and programmable data planes. He was a visiting researcher at Technion – Israel Institute of Technology and NOKIA Bell Labs.

Gianni Antichi is an Associate Professor at Politecnico di Milano and a Senior Lecturer (Associate Professor) at Queen Mary University of London. In the past he has worked as Senior Research Associate at the University of Cambridge and he received the M.Eng. and Ph.D. degrees in information engineering from the University of Pisa in 2007 and 2011, respectively. His research interests cover a broad spectrum of topics in both networks and systems, ranging from end-host network stacks to programmable switching architectures.

Pedro Henrique Gomes is a senior researcher at Ericsson Research, engaged in management of 5G networks and services. He is a delegate in the ETSI Zero-Touch Network & Service Management working group, contributing to the architecture definition especially with closed-loop automation and intent-driven enablers. He received a Ph.D. (2019) and M.Sc. (2015) in electrical engineering from the University of Southern California, Los Angeles, USA, and M.Sc. (2011) in computer science from the University of Campinas, Brazil.

Gergely Pongracz graduated from the Technical University of Budapest in 2000. In 2004, he became a Research Engineer at Ericsson Research, where he is an expert in researching programmable data plane. He is involved in NFV and SDN topics, especially in the programmable networking area. These projects resulted in well received papers and demos, such as a paper on the IEEE SigComm in 2016 or demos at the Mobile World Congress in 2015 and 2017.