

AccuPIPE: Accurate Heavy Flow Detection in the Data Plane Using Programmable Switches

Yang Guo
NIST

Franklin Liu
UIUC

An Wang
Case Western Reserve University

Hang Liu
The Catholic University of America

Abstract—Identifying heavy flows, i.e., flows with large packet counts during a pre-defined time window, is vital for many network applications. The task of real-time heavy flow detection in data plane is challenging due to high switching speed (100 Gbps), a large number of concurrent flows (millions of concurrent flows), and small memory footprint requirement. In this paper, we dissect the key factors that affect the existing detection scheme's accuracy, and propose AccuPipe, a new detection scheme with intelligent flow entry replacement strategies. The simulation results show that the new scheme is able to efficiently utilize all flow entries in the detection pipeline, and detects more than 850 heavy flows (out of top 1,000) using a small amount of memory (1,000 flow entries, roughly equivalently to 18KB memory) with reasonable reporting overhead. This represents a 76% improvement over HashPIPE scheme, which detects on average 484 heavy flows (out of top 1,000) in the same setting. In addition, we investigate the performance of different flow entry replacement strategies, and report their pros and cons.

I. INTRODUCTION

In the Internet, a small set of heavy flows, flows with extremely large number of packets or bytes, often accounts for a disproportionate share of the total traffic. Real-time detection of such heavy flows at small time scales is useful for many applications, e.g., DDoS detection, dynamic traffic routing [14], dynamic flow scheduling [15], etc. Real-time heavy flow detection in the data plane, however, is challenging due to stringent speed, accuracy, and memory requirement.

Heavy flow detection is well studied in streaming algorithms literature [6], where packets are processed as they pass through the measurement point. Packet sampling [5], [8], sketching [7], [9], [17], and counter-based algorithms [11] are representative techniques that tackle the trade-off between measurement accuracy, speed, and memory usage. For instance, sample and hold [8] improves the sampling accuracy by keeping counters for flows that have been sampled. Count-min sketch [7] hashes on packet headers and increments counters in hash tables. The minimum counter is used to approximate the flow packet counts. FlowRadar [9] improves upon count-min sketch by allowing keys (flow IDs) to be decoded from the hash. More recently, UnivMon [10] and Elastic Sketch [18] develop sketch based techniques where one sketch satisfies multiple tasks' requirements.

HashPipe [16] is the first to investigate the real-time heavy flow detection problem in a switch's data-plane. HashPipe takes advantage of emerging programmable switch's programmability [1]–[3] and designs a measurement pipeline

that implements modified space saving [12] algorithm to capture heavy flows. Elastic sketch [18] is another in-data-plane flow measurement scheme that utilizes more memories than HashPipe but provides the flow rate estimations for all flows.

In this paper we strive to improve upon HashPipe to achieve better heavy flow detection accuracy with small memory footprint. We analyze two key factors that affect HashPipe's measurement accuracy, namely *flow entry wastage* in the measurement pipeline and *packet arrival pattern*. Our analysis and simulation results show that the flow entry wastage contributes less than 8% to the accuracy loss, while the packet arrival pattern plays a major role in the measurement inaccuracy due to dramatic time-varying packet arrival pattern at fine time scale.

We propose AccuPipe that integrates frequency-based caching replacement strategies into heavy flow measurement pipeline. Instead of using the accumulated packet count as the indicator of potential heavy flows as in HashPipe, *AccuPipe utilizes flow entries inside the measurement pipeline to opportunistically capture short-term packet bursts and reports them to the measurement server*. At the end of each measurement cycle, the measurement server aggregates the reported packet counts to identify the top heavy flows. Our results show that the proposed scheme improves the identification accuracy from 48% to more than 85% with reasonable reporting overhead.

The rest of the paper is organized as follows. Section II introduces HashPipe and analyzes key factors that affect HashPipe's measurement accuracy. Section III describes AccuPipe and flow entry replacement strategies. Section IV presents the performance evaluation results. Section V concludes the paper.

II. HASHPIPE AND ITS ACCURACY ANALYSIS

HashPipe uses the measurement pipeline that implements modified space saving [12] algorithm to capture heavy flows. The measurement pipeline is defined using the programming language, e.g., P4 [2]¹ and runs in a programmable switch's data-plane [1]. The pipeline consists of multiple stages, with each stage working as a hash table. The entries in the hash

¹Certain commercial equipment, instruments, or materials are identified in this paper in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

table are called *flow entries*. In HashPipe, a flow entry contains two fields, *key* field and *val* field. The *key* field holds the flow id, e.g., five tuple $\{\text{srcIP}, \text{dstIP}, \text{srcPORT}, \text{dstPORT}, \text{Protocol}\}$ as defined by the measurement. The *val* field holds the accumulated packet count of the corresponding flow. Programmable switches use *packet metadata* to communicate the results of packet processing between different stages. The metadata traverses the pipeline together with the packet. In HashPipe, the packet metadata also contains two fields, *cKey* (*carried Key*) and *cVal* (*carried value*). The first stage hashes on the key of incoming packet (*iKey*), while the rest of stages hash on *cKey*.

The first stage in HashPipe always allows the incoming packet to be inserted into the hash table. Let $h_i(\cdot)$ be the hash function at stage i . The incoming packet is hashed to flow entry l , $l = h_i(iKey)$. If the entry l is empty and not used, we set $key_l = iKey$ and $val_l = 1$. The packet finishes the measurement and exits the pipeline. If key_l matches *iKey*, the value of val_l is increased by one and the packet also exits the pipeline. If *iKey* does not match key_l , $(cKey, cVal) \leftarrow (key_l, val_l)$ and $(key_l, val_l) \leftarrow (iKey, 1)$. The packet carries the metadata and goes to the next stage.

The rest of the stages keep heavy flows in the pipeline and push out small flows. Upon the arrival of a packet, the stage i hashes on *cKey* and gets flow entry l , where $l = h_i(cKey)$. If the flow entry l is empty or $key_l = cKey$, the entry is updated as in the first stage and the packet exits the pipeline. If, however, the key_l does not match *cKey*, *cVal* is compared with val_l . Whichever key with larger value will be kept in the hash table, and the smaller one is saved in the metadata and moved to the next stage. In the end the flow with the smallest *val* is washed out of HashPipe. The detailed algorithm is included in the Section 3.4 in [16].

A. Accuracy analysis of HashPipe

Flow entry wastage, traffic arrival pattern, and duplicates negatively affect HashPipe's accuracy. Consider a N flow-entry, K stage HashPipe pipeline. We define *heavy flows* to be the flows ranked in the top N based on their packet counts during a pre-defined time window. The rest of flows are called *light flows*. Ideally HashPipe should be able to capture all heavy flows, with one flow entry for one heavy flow. However, since flows are hashed randomly to flow entries, some flow entries may not "see" any heavy flows, and are thus wasted. We denote such behavior as *flow entry wastage*. Also, flow entries can be occupied by light flows instead of heavy flows. A light flow can overtake competing heavy flows with favorable *traffic arrival pattern*, as shown later in this section. Finally, a heavy flow may be hosted at multiple flow entries at different stages. Such *duplicates* also negatively affect HashPipe's accuracy. The simulation study in [16] shows that duplicates account for from 5% to 15% towards inaccuracy. In this section we focus on the effects of *flow entry wastage* and *traffic arrival pattern*.

- **Effect of flow entry wastage.** Below we derive a model that estimates S , the average number of wasted flow entries.

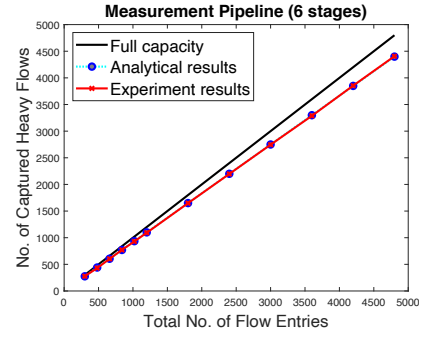


Fig. 1: HashPipe hash collision error analysis.

In the derivation, we ignore the *duplicates* effect and *traffic arrival pattern* effect. We also assume that the first HashPipe stage behaves the same as the later stages. These assumptions favor HashPipe and the analysis offers a lower bound of S .

Given N flow entries and K stages, the average number of flow entries per stage is N/K . Denote by s_i the average number of wasted entries at stage i , and by n_i the average number of incoming heavy flows that are looking to be hosted at stage i . We have $n_1 = N$, and

$$n_i = n_{i-1} - N/K + s_{i-1} \quad (1)$$

for $i = 2, \dots, K$. Stage $(i-1)$ has N/K entries with s_{i-1} being wasted. Hence the average number of heavy flows captured at stage $(i-1)$ is $N/K - s_{i-1}$. The number of incoming heavy flows at stage i is thus $n_{i-1} - N/K + s_{i-1}$.

We next derive the formula for s_i . Consider a heavy flow that is randomly hashed to a flow entry at stage i . The probability that a flow entry is not chosen by this heavy flow is $1 - \frac{1}{N/K}$. The probability that a flow entry is not chosen by any n_i heavy flows is $(1 - \frac{1}{N/K})^{n_i}$. Hence the average number of wasted entries at stage i is:

$$s_i = (N/K) \cdot (1 - \frac{1}{N/K})^{n_i} \quad (2)$$

for $i = 1, 2, \dots, K$. Starting with $n_1 = N$, $(\{s_i\}, \{n_i\})$ can be computed iteratively using Eqn.(1) and (2).

Denote by C the average number of captured heavy flows by HashPipe, i.e., $C = N - \sum_{i=1}^K s_i$. Figure 1 depicts the average number of captured heavy flows with varying number of total flow entries. The number of stages is set to be six, shown to be optimal in [16]. Varying the number of stages does not change our conclusions. We also conduct the simulations to verify the analytic model. We use a two-second CAIDA trace [4] with one million packets and over 100k flows to drive the simulation. Five-tuple $\{\text{srcIP}, \text{dstIP}, \text{srcPORT}, \text{dstPORT}, \text{Protocol}\}$ is used as the flow id through out the paper. The simulation results are consistent with the analysis. The HashPipe's measurement pipeline wastes no more than 8% of the flow entries. Using the same trace with 1,000 flow entries, HashPipe only captures about 484 heavy flows, or 48% of top 1,000 heavy-flows. This raises the question if *traffic arrival pattern* plays a more important role than *flow entry wastage* and *duplicates*.

- **Effect of traffic arrival pattern.** We next examine the impact of traffic arrival pattern to the HashPipe accuracy.

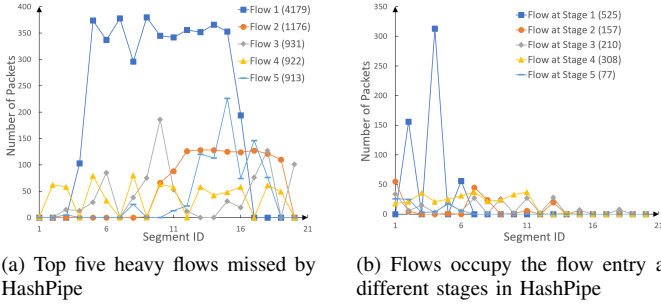


Fig. 2: Effect of traffic arrival pattern to the HashPipe accuracy.

To observe the traffic arrival pattern, the two-second trace is divided into 20 segments (0.1 second per segment). Fig. 2a plots the number of packet arrivals per segment for the top five heavy flows missed by HashPipe. The flows are ranked from one to five in the decreasing order of their packet counts, which is included in the parenthesis. Furthermore, Fig. 2b depicts the flows that occupy the flow entries that Flow 1 is hashed to at five pipeline stages (we ignore HashPipe's first stage since it is used for staging the measurement). Flow 1, with the packet count of 4,179, is a much heavier flow than any of the five flows hosted in HashPipe. Flow 1, however, does not arrive until 4th segment, when the other flows already have established a healthy packet count in the pipeline. Flow 1 fails to accumulate a big packet count in the initial stage of HashPipe and loses the competition with smaller flows. We also examine other heavy flows that are not captured by the HashPipe. The results show that the majority of the missed heavy flows are caused by the traffic arrival pattern instead of flow entry wastage and duplicates. It has long been observed that the network traffic is very bursty due to underlying protocol such as TCP, and packet burst varies dramatically at small time scale. A large packet count is not a robust indicator that a flow is heavy. The design of HashPipe favors flows with early burst packet arrival, not necessarily heavy flows.

III. DESIGN OF ACCUPIPE

In this section, we introduce *AccuPipe* (*Accurate HashPipe*) to accurately capture heavy flows. Unlike HashPipe where packet count is used to predict the heavy flows, AccuPipe uses the measurement pipeline as a cache to opportunistically capture ongoing heavy flows. When an ongoing flow is deemed not heavy anymore, its packet count is evicted from the measurement pipeline and reported to the server. The emptied flow entry continues to capture next heavy flow burst. At the end of a measurement window, the server aggregates collected packet counts and obtains top heavy flows.

In Caching technology, contents are stored in temporary storage, so-called cache, to reduce the service delay or the workload on servers [13]. To maximize the caching hit ratio, various cache replacement strategies have been developed, e.g., recency-based strategies (e.g. LRU), frequency-based strategies (e.g. LFU), function-based strategies, randomized

strategies, etc. AccuPipe use the measurement pipeline as a cache to capture the heavy flow packet burst. A set of flows being hashed to the same flow entry compete for its occupancy; replacement strategies are designed to store heavy flows with frequent packet arrivals. Hence frequency-based caching strategies are most suitable for our design. Below we describe several replacement strategies used in AccuPipe.

•**Aging based strategy:** An *age* field is added to flow entries. Upon the arrival of a packet, if *iKey* matches flow entry *key*, the packet count is increased by one and the *age* field is reset to be zero, regardless of the current value. The packet then exits the pipeline. If the packet *iKey* does not match flow entry's *key*, the *age* field is increased by one. The *age* is then compared to a pre-set threshold α . If the *age* is less than α , the packet continues to traverse the pipeline. On the other hand, if the *age* is greater than *alpha*, the current flow in the flow entry is deemed to be infrequent and evicted. Its packet count is reported to the server. The newly arrived packet is cached into the flow entry and exits the pipeline.

•**Frequency based strategy:** Frequency based strategy was first proposed in [18]. In frequency based strategy, a new field called *total_packet_count* is added to each flow entry. The *total_packet_count* is initialized to be zero, and increased by one whenever a packet is hashed to this flow entry. The ratio of *total_packet_count* / *packet_count* is computed and compared to the preset threshold λ . The rest of the process is the same as in the Aging based strategy.

•**Segment based strategy:** In segment based strategy, AccuPipe works the same as HashPipe. The measurement time window is divided into equal size small measurement sub-windows, or segment. At the end of each segment, the measurement results in the pipeline are reported to the server. The pipeline is re-initialized and continues the measurement for the next segment.

•**Hybrid strategy:** Hybrid strategy is a combined scheme of aging based strategy and frequency based scheme. If the packet count is less than or equal to the pre-set threshold β , frequency based strategy is applied. If the packet count surpasses β , the aging based strategy is applied. As shown in Section IV, hybrid strategy offers best performance.

IV. EVALUATIONS

In this section, we evaluate AccuPipe with different caching strategies. The measurement pipeline consists of six stages and 1,000 flow entries. The pcap trace is from [4] and is pre-processed to extract ten 5-second traces based on packet time-stamp information. Each 5-second trace contains more than 2.5 million packets and over 200K flows. The results presented here are the average over ten traces. Two metrics are examined: *accuracy*, the number of heavy flows that are correctly identified, and *reporting overhead*, the amount of reports that are sent to the server. Note that heavy flows are the flows being ranked in the top 1,000.

•**Performance comparison of Aging, Frequency, and Segmented AccuPipe.** The aging threshold, frequency threshold, and number of segments are varied in evaluating the

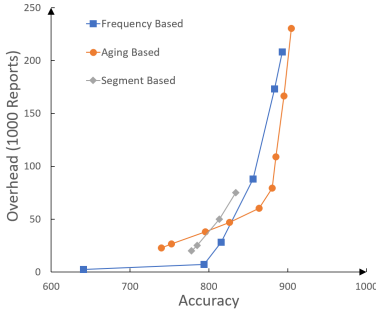


Fig. 3: Performance comparison of Aging, Frequency, and Segmented AccuPipe. Each dot represents the average results over ten traces for one experiment setting. The aging thresholds (α) from the right to the left are 8, 16, 32, 48, 64, 80, 96, 128, and 144. The frequency thresholds (λ) from the right to the left are 2, 4, 8, 16, 32, and 64. In Segmented AccuPipe experiments, the number of segments are 20, 25, 50, and 75, respectively from the left to the right.

accuracy and the reporting overhead of Aging AccuPipe, Frequency AccuPipe, and Segmented AccuPipe. Fig. 3 depicts the reporting overhead vs. accuracy with different thresholds and number of segments, where accuracy represents the number of heavy flows that are correctly identified. As expected, smaller aging and frequency thresholds offer higher accuracy than larger ones as AccuPipe captures smaller traffic bursts and report more frequently. Aging scheme can capture more than 850 heavy flows accurately as long as the aging threshold is smaller than 64, while Frequency scheme can do so if the frequency threshold is no greater than eight. The reporting overhead for Aging and Frequency AccuPipe are 60.1k and 87.9k reports, respectively, which favors aging scheme.

However, as the threshold decreases, frequency based scheme starts to outperform aging scheme. At the frequency threshold of 32, the *accuracy percentage*, the percentage of heavy flows that are correctly identified, reaches 79.3% with only 7.1k reports in Frequency AccuPipe; the similar accuracy percentage (79.5% with the aging threshold of 96) requires 60.1k reports in Aging AccuPipe. In general, aging based scheme performs well in high accuracy percentage ($> 85\%$) region while frequency scheme performs better in medium accuracy region, as shown in Fig. 3.

In Segmented AccuPipe, the packet counts in the pipeline are reported to the switch controller at the end of each segment, and the measurement is restarted for the next segment. We assume that each flow entry requires one report. In general, using more segments in Segmented AccuPipe leads to higher accuracy since the measurement is able to keep track of traffic pattern at the finer time scale. In terms of reporting overhead, Segmented AccuPipe incurs more overhead than Frequency AccuPipe to achieve the same accuracy. Compared to Aging AccuPipe, Segmented AccuPipe incurs more overhead when the accuracy is greater than 800 heavy flows. It, however, starts to outperform Aging AccuPipe as the accuracy decreases to be smaller than 800 heavy flows. Note that the packet counts in Segmented AccuPipe are periodically reported. Hence these

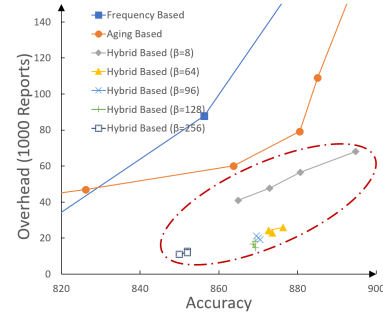


Fig. 4: Performance of Hybrid AccuPipe. The value of α is set at 32 for all cases. The value of β is shown in the figure. The value of λ are set at 32, 64, 96, and 128 from the right to the left. We retain the results of Aging and Frequency AccuPipe for the purpose of comparison.

reports form a batch and can potentially be compressed. Compression can drastically reduce the reporting overhead and make Segmented scheme desirable.

• **Performance of Hybrid AccuPipe.** Compared to Frequency AccuPipe, Aging AccuPipe is more accurate and requires less reporting overhead in high accuracy region (accuracy percentage $> 85\%$). Frequency AccuPipe, however, is able to bring down the reporting overhead without significantly sacrificing the accuracy in lower accuracy region. Hybrid AccuPipe strives to combines Aging scheme's accuracy with Frequency scheme's reporting efficiency. In the Frequency AccuPipe with the threshold of λ , a flow with the burst arrival of n packets at the beginning can occupy the flow entry for the period of $\lambda \cdot n$ packet arrivals hashed to the same flow entry. A large value of λ could casues the overstay of unheavy flows. In contrast, Aging AccuPipe always examines the most recent α packets. If no packet belonging to the flow entry key arrives during the most recent α packet arrivals, the packet count in the flow entry will be evicted.

To avoid the overstay of a unheavy flows in Frequency AccuPipe, a new threshold β is introduced in Hybrid AccuPipe. Hybrid AccuPipe has three parameters, α , β , and λ . When a flow entry's packet count is less than β , AccuPipe employs Frequency scheme. Otherwise AccuPipe employs Aging scheme. Fig. 4 depicts the accuracy vs. overhead of Hybrid AccuPipe with different values of β . Hybrid AccuPipe consistently outperforms Aging and Frequency AccuPipe. For instance, when $\beta = 64$, Hybrid AccuPipe achieves 87.3% accuracy percentage with 22.9k reports.

V. CONCLUSIONS

In this paper, we study the accuracy issue of the real-time heavy flow detection using programmable switch. We investigate two factors that contribute to the measurement inaccuracy, and identify traffic arrival pattern is the main contributor to the inaccuracy in the existing scheme. We propose AccuPipe that employs caching techniques to capture short-term traffic bursts. The simulation results show that the AccuPipe improves the heavy flow detection rate by over 70% over the existing scheme.

REFERENCES

- [1] Barefoot Networks. Barefoot Tofino. <https://www.barefootnetworks.com/technology/>.
- [2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. "P4: Programming protocol-independent packet processors." In ACM SIGCOMM Computer Communication Review, 2014.
- [3] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN." In ACM SIGCOMM, 2013.
- [4] The CAIDA Anonymized Internet Traces 2016 Dataset. http://www.caida.org/data/passive/passive_2016_dataset.xml.
- [5] Cisco Networks. Netflow. <http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html>.
- [6] G. Cormode and M. Hadjieleftheriou. "Finding frequent items in data streams." In VLDB Endowment, 2008.
- [7] G. Cormode and S. Muthukrishnan. "An improved data stream summary: The count-min sketch and its applications." In Journal of Algorithms, 55(1):58–75, 2005.
- [8] C. Estan and G. Varghese. "New directions in traffic measurement and accounting." In ACM Trans. Computer Systems, 21(3), 2003.
- [9] Y. Li, R. Miao, C. Kim, and M. Yu. "FlowRadar: A better NetFlow for data centers." In USENIX NSDI, 2016.
- [10] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. "One sketch to rule them all: Rethinking network flow monitoring with univmon." In Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference. ACM, 2016.
- [11] G. S. Manku and R. Motwani. "Approximate frequency counts over data streams." In VLDB Endowment, 2002.
- [12] 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. Springer, 2005.
- [13] Stefan Podlipnig and Laszlo Boszormenyi. "A Survey of Web Cache Replacement Strategies", ACM Computing Surveys, 2003.
- [14] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca. "Planck: Millisecond-scale monitoring and control for commodity networks." In ACM SIGCOMM, 2014.
- [15] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown. "Programmable packet scheduling at line rate." In ACM SIGCOMM, 2016.
- [16] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S. Muthukrishnan, and Jennifer Rexford. "Heavy-Hitter Detection Entirely in the Data Plane." In SOSR 2017.
- [17] R. Schweller, A. Gupta, E. Parsons, and Y. Chen. "Reversible sketches for efficient and accurate change detection over network data streams." In ACM IMC, 2004.
- [18] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. "Elastic Sketch: Adaptive and Fast Network-wide Measurements", In SIGCOMM 2018.