PNT Sketch: A Generic Sketch Algorithm for Periodic Network Telemetry

Quan Yuan*, Fenghua Li*, Jessie Hui Wang*, Shuanghong Yu*, Allen Hong*, Xingkun Yao†
*Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University, Beijing, China
†Information Technology Center, Tsinghua University, Beijing, China

***Abstract*—Network telemetry provides operators with a fine-grained internal view of network behavior. Recently, various lightweight and high-accuracy sketch algorithms have been proposed for real-time full flow record collection. However, they still have at least two deficiencies: (1) they are mostly designed for counting and cannot accurately support other kinds of aggregation functions, such as max and mean; and (2) the vast majority of them are focused on improving the measurement accuracy of each individual period, overlooking the fact that many elephant flows last for multiple periods. In this paper, we propose PNT sketch, a generic algorithm with well-designed data structures and strategies. The PNT sketch implements multiple aggregation functions to record flow attributes for elephant flows and improves the accuracy of a new period through the full use of historical flow records collected in the previous period with our history-aware reset strategy. We conduct a theoretical analysis of our algorithm, and the results show that it has a more rigorous error bound than the Elastic sketch. We implement our algorithm on both the x86 and P4 platforms and evaluate its performance on seven different tasks. Compared to the state-of-the-art, the PNT sketch achieves 1.4 $\sim$ 437.2 smaller error rates.**

***Index Terms*—Network measurement; Telemetry; Generic; PNT; Sketch**

I. Introduction

Network measurements provide essential and critical information for modern network management. With the continuous expansion of network scale and the emergence of new network applications, such as self-driving and application-aware networks, traditional sampling-based measurement methods, such as SNMP [1] and NetFlow [2], can no longer meet increasingly fine-grained measurement requirements and are gradually being replaced by network telemetry. Benefiting from active reporting, the telemetry method only takes up a small number of CPU cycles and can achieve a second or even sub-second reporting timescale. However, due to the resource limitations of forwarding devices, established methods cannot record all flows. For example, AM-PM [12] only supports recording a small number of marked flows. Recently, sketch-based algorithms [3-11,14-18] have become the preferred solution to this problem.

Through the design of data structures and strategies, sketch-based algorithms can record a large number of flow attributes in a limited space and achieve a trade-off between high accuracy and low overhead. But existing algorithms still have at least two deficiencies.

ISBN 978-3-903176-48-5

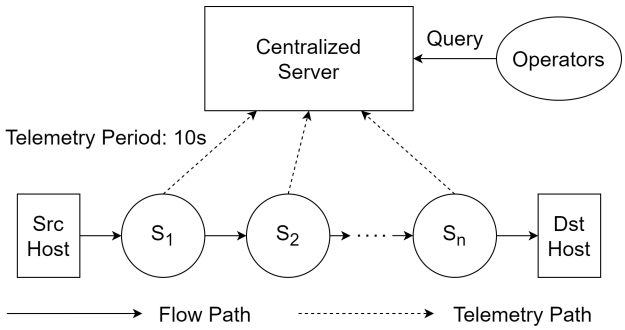


Fig. 1: The workflow of out-band network telemetry: intermediate devices monitor packet attributes and aggregate them according to the flow level. Then all records are periodically reported to the centralized server for analysis.

First, they are mostly designed for counting and cannot accurately support other kinds of aggregation functions [22], such as max and mean. This is because these algorithms suffer from hash collisions, and for a certain attribute, only a part of the measured flows can be accurately recorded with limited resources. For example, for flow size estimation, errors have little effect on elephant flows [21] because of their large cardinality. But for maximum delay estimation, a mouse flow with a high delay has a small estimation error and introduces large errors in the estimation of its colliding elephant flows. Fine-grained network management requires the support of multiple accurate measurement metrics for elephant flows and needs a generic algorithm to record them.

Second, the vast majority of them are focused on improving the measurement accuracy of each individual period by improving data structures and strategies. For example, [4-6] optimize update and query strategies, and [7-9,14,16-17] design their own data structures. But they overlook the fact that many elephant flows last for multiple periods. If we can reserve space for potential elephant flows that may occur in a new period in advance based on historical records collected from the previous period, there is no error for them. But, in existing algorithms, all records are reset directly at the start of each period.

In this paper, we design and implement PNT sketch, which is a generic sketch algorithm for periodic network telemetry and achieves high-efficiency updates and high-accuracy query results. We follow the direction proposed by [7-9] to record

flows of different sizes separately and go a step further in improving generality and accuracy. Specifically, our algorithm has the following contributions:

- With well-designed data structures and strategies, the PNT sketch implements multiple accurate aggregation functions (including sum, max, and mean) to record flow attributes for elephant flows.
- Our detailed analysis of multiple traffic traces shows that about 80% of elephant flows are the same for two adjacent measurement periods. Then we craft a history-aware reset strategy to improve the accuracy of a new period through the full use of historical records collected in the previous period.
- We conduct a theoretical analysis of our algorithm. The results effectively guide the parameter selection of the PNT sketch and show that it has a more rigorous error bound than the Count-min sketch and the Elastic sketch.
- We implement our algorithm on both the x86 and P4 platforms and evaluate its performance against four state-of-the-art algorithms on seven different measurement tasks. Compared to the state-of-the-art, the PNT sketch achieves 1.4 $\sim$ 437.2 smaller error rates.

The remainder of the paper is organized as follows. We first review some sketch-based network measurement algorithms in Section II and then introduce the rationale behind our algorithm design in Section III. We present algorithm details of the PNT sketch in Section IV and give a corresponding theoretical analysis in Section V. Then we describe the implementation details of the PNT sketch and four state-of-the-art competitors in Section VI and evaluate their performance in Section VII. Finally, we conclude the paper in Section VIII.

II. Related Work

A. Sketch Algorithms for Counting

Sketch-based algorithms were originally designed for counting. Count-Min sketch (CM) [3] is the most well-known sketch algorithm used to count flow attributes (e.g., packet number and packet size). It consists of d arrays. Each array has w counters and an independent hash function that is used to map a flow uniformly and randomly into one of its counters. When a packet of flow f arrives, the CM sketch finds all counters that flow f is mapped to by hashing and increments them by 1 (for counting packet number). When querying a flow attribute, the CM sketch returns the minimum value among all mapped counters. The CM sketch has over-estimation errors because of hash collisions between different flows.

Subsequent works focus on improving the data structure and strategies of the CM sketch to get higher accuracy. Count sketch [4], CU sketch [5], and Count-Mean-Min sketch [6] all use the same data structure as the CM sketch, but optimize update and query strategies. In particular, the CU sketch proposes a strategy called "conservative update." For each update, only the smallest one or several counters are incremented, thereby minimizing the impact of collisions between different flows.

Furthermore, Elastic sketch [7], Augmented sketch [8], and HashFlow [9] design their data structures and strategies to record elephant flows and mouse flows in different data structures, which can be called the heavy part and the light part. In their heavy parts, each counter has a one-to-one correspondence with a unique flow. Therefore, if a flow is recorded in the heavy part, there are no errors in subsequent updates. Because we cannot know which flow is an elephant flow until enough packets of it are recorded, these algorithms all design a replacement strategy to swap elephant flows into their heavy parts and mouse flows out. This replacement process is frequently triggered at the beginning of each measurement period and introduces errors in the heavy part. In modern network telemetry, a period length is usually on the order of seconds, such as 10 s. Therefore, the errors introduced by the replacement process are magnified.

SuperFlow [10] realizes continuous automatic export of flow records. Its auxiliary table A can identify inactive flows and then export their records to the collector, freeing up space for recording new flows. But in this way, we cannot know the overall network traffic information for a certain period because the SuperFlow does not periodically export records for unfinished flows.

B. Sketch Algorithms for Other Tasks

At present, there are only a few works trying to use sketches for measurement tasks other than counting, such as recording the maximum delay of any flow.

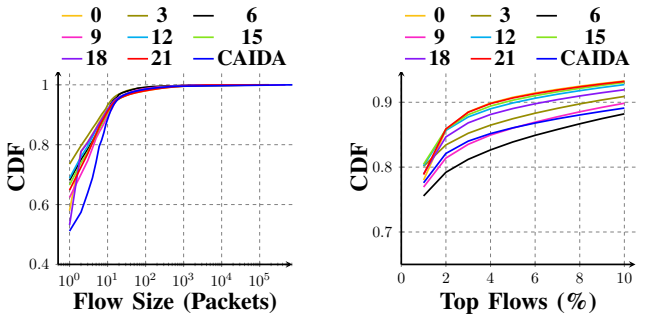
In order to support recording multiple attributes, SuMax sketch [11] adds max cells based on the CM sketch and combines multiple sum cells and max cells to form a bucket, which is used to record multiple attributes of a flow. For the update of sum cell, the SuMax sketch simplifies the conservative update strategy of the CU sketch to run on resource-limited P4 switches. But for max cell, the SuMax sketch only records the maximum value between the original value and the new attribute value. Therefore, the ARE (average relative error) of estimated values is often greater than 10, which can only be used to identify extreme network anomalies and not for fine-grained network optimization algorithms.

There are also some dedicated algorithms designed for certain tasks. For example, LDS [15] records the average delay for any flow, and Zhu et al. [18] detect flows with high Intra-Flow Packet Delay (IFPD). But they all have their own designs, and resource-limited network devices cannot support running multiple algorithms at the same time.

III. Motivation

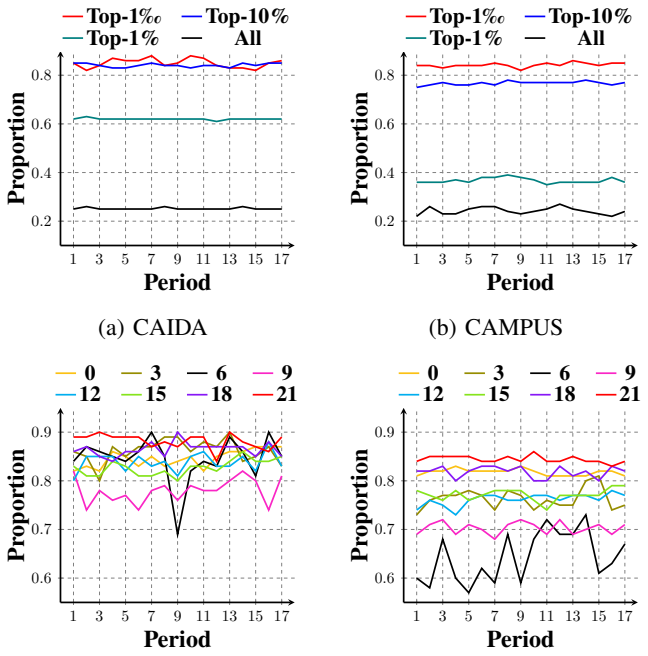
A. Trace Analysis

Due to the effects of human habits of work and rest, the network traffic patterns change dramatically over time, making it impossible to obtain comprehensive conclusions by analyzing only one traffic trace captured at a single moment. Starting at 0 a.m. on a workday, we collect a 3-minute trace every 3 hours. And we also introduce another 3-minute trace from the CAIDA 2018 [13] to analyze together. Table I shows the differences between different traces, where $CAMPUS_t$ means that the trace was collected at moment t. It can be



(a) Flow Size Distribution (b) Packet Distribution

Fig. 2: Mouse flows account for a great portion of all flows, but most packets are from elephant flows.



(a) CAIDA (b) CAMPUS

(c) Top-1‰ (d) Top-1%

Fig. 3: Proportion of the same flows between two adjacent measurement periods.

observed that the flow number and the average flow size fluctuate with time, with the peak at 9 p.m. This is because, after a full day at work, most people take a break and use more applications that generate elephant flows, such as videos and p2p downloads. Further investigation reveals that different traces exhibit similar long-tail effect characteristics: almost 90% of flows are mouse flows with less than 10 packages (as shown in Fig. 2a), while the top 10% of flows include more than 85% of packets (as shown in Fig. 2b).

Elephant flows last for multiple periods. As shown in Fig. 3, we count the proportion of identical flows for two adjacent measurement periods with a period length of 10 s. The x-axis represents the index of each period. For example, the first point of the red line in Fig. 3a means that in the CAIDA trace, 85% of the largest one-thousandth flows (named top-1000 flows) that appear in period 1 are also part of the top-1000 flows of period 0. Then we can see that the proportion is fairly high for elephant flows. For both the top 1‰ and 1% of flows in the CAIDA trace, the proportions are around 85%. And in the CAMPUS trace, the proportions are around 76% and 65%, respectively. As a consequence, when a new period starts, we may assume that the next measured elephant flow set is the same as the one in the previous period, and then progressively fix the incorrect component. We also analyze the variation of the proportion at different moments. As shown in Fig. 3c-3d, the proportion is highest and most stable at 9 p.m. This is in line with our analysis of the traffic patterns above.

TABLE I: Trace information

Trace	Flow Number	Ave. Flow Size	Max Flow Size
$CAMPUS_0$	743504	23.44 pkts	261889 pkts
$CAMPUS_3$	577821	14.55 pkts	180597 pkts
$CAMPUS_6$	378191	13.79 pkts	81130 pkts
$CAMPUS_9$	568439	18.44 pkts	475961 pkts
$CAMPUS_{12}$	689553	21.12 pkts	169617 pkts
$CAMPUS_{15}$	781052	24.20 pkts	160909 pkts
$CAMPUS_{18}$	786582	20.38 pkts	148453 pkts
$CAMPUS_{21}$	896440	24.92 pkts	178341 pkts
$CAIDA$	3409768	23.33 pkts	722627 pkts

B. Aggregation Functions

Counting: For many measurement tasks, counting the number of packets is a useful method. First, we may calculate the count difference of a flow between any two neighboring switches to determine hop-by-hop packet loss. Second, we can track a flow's forwarding path by finding all devices where its count is not zero. Finally, additional common tasks include flow size estimation, heavy hitter detection, and heavy change detection. These tasks can assist network operators quickly spot network anomalies like DDoS attacks.

Max and mean: There are also many packet attributes that cannot be aggregated by counting, such as Intra-Flow Packet Delay (IFPD), which refers to the time interval between two adjacent packets of a flow reaching the same device. By accurately monitoring the maximum and average values of the IFPD hop by hop, we can detect aberrant jitters caused by queuing, congestion, or network attacks and locate them. Jitters can have a substantial impact on the quality of streaming media (e.g., audio, video, and music).

In summary, a generic measurement algorithm should support multiple aggregation functions, including **sum** (counting), **max**, and **mean**, while having high accuracy.

C. Basic Idea

Record multiple flow attributes for elephant flows: For network management, elephant flows are more important. For example, they are typically reported to the controller as traffic demands in TE projects. Then, TE algorithms like FFC [18] and TeaVaR [19] reserve enough bandwidth for them to ensure

their availability even in the event of a major network failure. In contrast, mouse flows are only sent through idle bandwidth as background traffic. Elephant flow scheduling quality has an impact on network service quality. Good flow scheduling requires the support of multiple accurate measurement metrics. We design our data structures and strategies to automatically identify flows of various sizes and record them in two sketches, the H part and the L part, respectively. And each bucket in the H part has multiple cells to record multiple attributes for elephant flows. In addition, the L part only records the flow size (usually the packet number) of remaining flows to save resources. In this paper, elephant flows refer to top-k flows, the k flows with the highest number of packets. To ensure that the vast majority of them can be recorded in the H part, we select parameters through theoretical analysis.

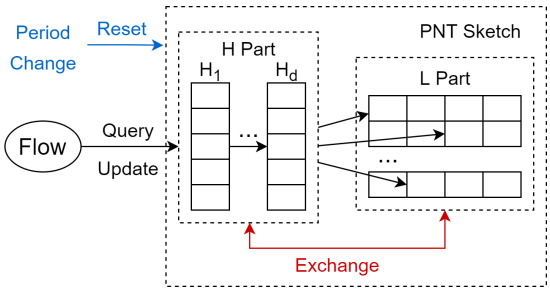
History-aware reset strategy: The error of an elephant flow can be avoided if the flow is recorded in the H part from the start. We propose a simple strategy that can effectively use historical records collected in the previous period to anticipate potential elephant flows in the current period in advance and reserve buckets for them, thus improving their accuracy. Specifically, buckets in the H part are not directly reset when a new period starts. We identify the top-k flows and mark their buckets. All marked buckets will not be preempted by other flows in the new period, thus ensuring that the top-k flows are recorded directly in the H part once they reappear. For other buckets, we reset them directly to record new flows.

IV. The Design of PNT Sketch

A. Overall Framework and Data Structures

As shown in Fig. 4a, the PNT sketch consists of two parts: the H part recording attributes from elephant flows and the L part recording attributes from others. The $H(L)$ part consists of $d_H(d_L)$ bucket arrays and $d_H(d_L)$ independent hash functions. Each array consists of $w_H(w_L)$ buckets. In practice, we set $d_H = d_L = d$ and use the same hash functions $h_1(\cdot), \cdots, h_d(\cdot)$ for both parts in order to reduce hashing overhead. When a new measurement period starts, we reset the PNT sketch using the history-aware reset strategy described in Section IV.E.

Update process: When a packet arrives, we extract its flow ID fid. Then we try to record it in the H part by looking for a bucket that records the same fid or an empty bucket. If we cannot find such a bucket, we record it in the L part. In the H part, each bucket records attributes from a uniquely determined flow, so it has high accuracy. In contrast, the L part uses a similar recording method as the CM sketch. A flow is recorded in multiple buckets, and each bucket may be shared with other different flows. In order to ensure that elephant flows are recorded as much as possible in the H part, we have designed an exchange strategy. When it is found that a flow in the L part is larger than a flow in the H part, their record positions are exchanged to make the larger one be recorded in the H part. Next, we present the details of each strategy and the pseudocode of the update process.



(a) The framework of PNT

FID	Num	Time	Max	Mean	Old

(b) The bucket structure of the H part

Fig. 4: The framework of PNT and bucket structures of H

Bucket structures and process: The bucket structure of the L part is the same as that of the CM Sketch, and each bucket contains a counter, which is used to record the number of packets. For the H part, to support various measurement tasks, we may need more than one kind of cell in each bucket. Different measurement tasks require different bucket structures. For example, in order to record Intra-Flow Packet Delay (IFPD) related attributes, we design the bucket structure (Fig. 4b) as follows:

- a fid cell recording the flow ID;
- a num cell (sum cell) recording the packet number of a flow;
- a $time$ cell (maximum cell) recording the arrival time of the last packet of a flow;
- a max cell (maximum cell) recording the maximum IFPD of a flow;
- a $mean$ cell (sum cell) recording the sum of IFPDs of a flow.
- an old cell recording the packet number of a flow before it is recorded in the H part, or identifying whether a flow is a potential elephant flow.

When a packet arrives and matches a bucket in the H part, we update the bucket according to the following steps. First, let t_{now} be the ingress timestamp of this packet. Second, we get the last arrival time, which is recorded in the time cell, and record it as t_{last}. Then we can calculate $delay$ = t_{now} - t_{last}. When t_{last} = 0, we consider the current packet as the first packet of a flow, and set $delay$ = 0. Finally, we update the flow record by adding one to the num cell, setting the $time$ cell to t_{now}, setting the max cell to the maximum of its old value and $delay$, and adding $delay$ to the $mean$ cell. Note that instead of recording the average IFPD directly, we record the sum of IFPDs and then calculate and return the average IFPD at query time.

B. Strategies

Update strategy: When a packet with flow ID f arrives, we first map it into the bucket $H_1[idx]$, where $idx = h_1(f)\%w_H$. If $H_1[idx]$ is empty, we initialize the bucket by setting it to

Algorithm 1: PNT Sketch: Update Strategy

Input: packet p

$f \leftarrow p.flowID, time \leftarrow timestamp, min_H \leftarrow \infty$
for $i = 1$ *to* d **do**
  $idx \leftarrow h_i(f)\%w_H$
  if $H_i[idx].fid ==$ "" **then**
    $H_i[idx] \leftarrow (f, 1, time, 0, 0, 0)$
    return
  else if $H_i[idx].fid == f$ **then**
    $delay \leftarrow time - H_i[idx].time$
    $H_i[idx].num++$
    $H_i[idx].time \leftarrow time$
    $H_i[idx].max \leftarrow max(H_i[idx].max, delay)$
    $H_i[idx].mean \mathrel{+}= delay$
    return
  else if $H_i[idx].old > 0$ **then**
    continue
  else if $min_H > H_i[idx].num$ **then**
    $min_H \leftarrow H_i[idx].num$
    $t \leftarrow i$, $idx_{min} \leftarrow idx$
$min_L \leftarrow \infty$
for $i = 1$ *to* d **do**
  $idx \leftarrow h_i(f)\%w_L$
  if $L_i[idx] < min_L$ **then**
    $min_L, L_i[idx] \leftarrow L_i[idx] + 1$
if $min_L > min_H$ **then**
  $tmp \leftarrow H_t[idx_{min}]$
  $H_t[idx_{min}] \leftarrow (f, min_L, time, 0, 0, -min_L)$
  for $i = 1$ *to* d **do**
    $idx \leftarrow h_i(tmp.fid)\%w_L$
    if $L_i[idx] < tmp.num$ **then**
      $L_i[idx] \leftarrow tmp.num$

$(f, 1, time, 0, 0, 0)$, where $time$ is the ingress timestamp. If $H_1[idx].fid = f$, the bucket is already occupied by the flow f, then we update it using the methods described in the previous subsection. In either case, we find a proper bucket to record the packet, so the update process finishes. Otherwise, a collision occurs, then we repeat the same process on $H_2, \cdots, H_d$ one by one. And in the meantime, we need to record the minimum value min_H of sum cells in all mapped buckets and its position (t, idx_{min}) for use in the exchange process.

If we cannot find a proper bucket, the packet needs to be recorded in the L part. We adopt a conservative update strategy. First, we set min_L to ∞. Then we iterate through each of the f-mapped buckets. If $L_i[idx]$ is less than min_L, we update min_L and $L_i[idx]$ to $L_i[idx] + 1$. Otherwise, we keep them unchanged. Finally, min_L is the current estimate of flow f size. Based on this strategy, the L part can achieve a performance close to that of the CU sketch.

Exchange strategy: If min_L is greater than min_H, the exchange process is triggered. We record the larger flow in the H part and the swap-out flow in the L part. First, we temporarily record the attributes of the swap-out flow in tmp. Then, we set $H_t[idx_{min}]$ to $(f, min_L, time, 0, 0, -min_L)$. $-min_L$ represents the packet number of the swap-in flow before it is recorded into the H part. Finally, we use another conservative update strategy to record the swap-out flow into the L part. Specifically, for all mapped buckets with values less than $tmp.num$, we update them to $tmp.num$ and leave all other buckets unchanged.

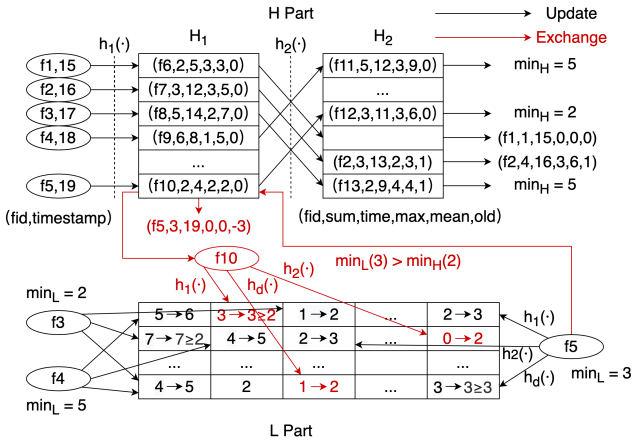


Fig. 5: Examples of PNT.

Query strategy: When a query arrives, we first look up the H part to try to find a bucket b with the same fid. If found, we return the attributes recorded in the bucket. For average IFPD, we return $\frac{b.mean}{b.num+b.old}$ if $b.old < 0$, and $\frac{b.mean}{b.num-1}$ otherwise. If there is no such bucket, the queried flow must be recorded in the L part. We return the minimum of all mapped buckets.

History-aware reset strategy: When a new measurement period starts, we choose a $threshold$ and iterate through all the buckets in the H part. If the num cell value of a bucket is greater than $threshold$, we keep its fid cell, set its old cell to 1, and set its other cells to 0. Then we simply reset the other buckets in both parts. In our software implementation, $threshold$ is set to the k-th largest value of all num cells in the H part, where k is the same as that in "top-k flows". And in the P4 implementation, we set $threshold$ to a preset value and update it periodically.

In conclusion, compared to the Elastic sketch, our algorithm avoids some elephant flows being recorded in multiple buckets. Compared with the Augmented sketch, our algorithm is based on hash addressing and can be implemented on a P4 switch. And compared to the Hashflow, our algorithm records all flows and avoids under-estimation errors. Then our algorithm improves accuracy for both elephant and mouse flows by reducing the impact of hash collisions with two conservative update strategies and reserving buckets for elephant flows with the history-aware reset strategy.

C. Examples

We use a few examples to illustrate the above strategies, and plot the workflow in Fig. 5. To simplify the process for ease

of understanding, we use $d_H = 2$. When a packet of flow f_1 arrives, it is first mapped into $H_1[h_1(f_1)]$, where it collides with f_6. Second, it is mapped into $H_2[h_2(f_1)]$ which is an empty bucket, and then the content of the bucket becomes $(f_1, 1, 15, 0, 0, 0)$, where 15 is the ingress timestamp of the packet. When a packet of f_2 arrives, it collides with f_7 in H_1, but is matched with $H_2[h_2(f_2)]$ which has the same fid. So $delay = 16\ (timestamp) - 13\ (H_2[h_2(f_2)].time) = 3$, the num cell is simply incremented by 1, the $time$ cell is set to 16, the max cell is set to $max(2, 3)$ and the $mean$ cell is incremented by 3. When a packet of flow f_3, f_4 or f_5 arrives, it collides with other flows both in H_1 and H_2, then it is mapped to the L part. In the update process, we calculate min_H and min_L. For flow f_4, $min_L = min(6, 6, 5) = 5 \leqslant min_H = min(6, 5) = 5$, the exchange process is not triggered. For flow f_3, min_H is also not less than min_L, but note that for the record of f_{13}, its old cell is equal to 1, so when calculating min_H we ignore this record. Then the min_H of flow f_3 is just 5 (the num cell value of f_8). For flow f_5, $min_L = 3 > min_H = 2$, so we record $(f_5, 3, 19, 0, 0, 0)$ in $H_1[h_1(f_5)]$ and map f_{10} to the L part. The sum cell value of f_{10} is 2, so all mapped buckets with values less than 2 are set to 2.

V. ANALYSIS

A. Elephant Flow Collision Number

To ensure that the vast majority of the top-k flows can be recorded in the H part with minimal memory and hash calculation overhead, we need to choose appropriate d_H and w_H. So, for a given d_H and w_H, we calculate the expectation of the number of the top-k flows that cannot be recorded in the H part due to hash collisions. Suppose each hash function is a standard 2-universal hash function. Then, given an arbitrary flow and an arbitrary bucket, the probability that the flow is mapped to the bucket is $\frac{1}{w_H}$. Thus, for any bucket in the first array, the number of the top-k flows that are mapped to the bucket T follows a Binomial distribution $B(k, \frac{1}{w_H})$. When k is large (e.g., $k > 100$) and $\frac{1}{w_H}$ is a small probability, T approximately follows a Poisson distribution $\pi(\frac{k}{w_H})$, i.e.,

$$Pr\{T = i\} = e^{-\frac{k}{w_H}} \frac{(\frac{k}{w_H})^i}{i!} \tag{1}$$

then the expectation of the number of buckets that records a top-k flow in the first array is

$$E_{in}^1 = w_H(1 - Pr\{T = 0\}) = w_H(1 - e^{-\frac{k}{w_H}})$$

When a collision occurs, the largest flow is recorded in the bucket, and others are sent on to the following arrays. Therefore, the number of the top-k flows that need to be mapped in the second array is

$$E_{out}^1 = k - E_{in}^1 = k - w_H(1 - e^{-\frac{k}{w_H}})$$

and so on

$$E_{out}^2 = E_{out}^1 - w_H(1 - e^{-\frac{E_{out}^1}{w_H}})$$

$$\vdots$$

$$E_{Light} = E_{out}^{d_H} = E_{out}^{d_H - 1} - w_H(1 - e^{-\frac{E_{out}^{d_H - 1}}{w_H}})$$

The relationship between w_H, d_H, k, and E_{Light} is shown in table II. When $w_H = k$ and $d_H = 4$, $\frac{E_{Light}}{k} = 1.6 * 10^{-6}$. In our experiments, the number of elephant flows is generally fewer than 10,000. Let $k = 10,000$, then $E_{Light} = 0.016 < 1$. Therefore, in this paper, we set w_H to k and d_H to 4.

TABLE II: The relationship between w_H, b_H, k, and E_{Light}

E_{Leight}/k \ d_H / w_H/k	**1**	**2**	**3**	**4**	**5**
0.5	0.56	0.22	$4.5 * 10^{-2}$	$2.0 * 10^{-3}$	$3.9 * 10^{-6}$
1	0.36	0.06	$1.8 * 10^{-3}$	$1.6 * 10^{-6}$	$1.2 * 10^{-12}$
1.5	0.27	0.02	$1.7 * 10^{-4}$	$1.0 * 10^{-8}$	$5.3 * 10^{-18}$
2	0.21	0.01	$3.0 * 10^{-5}$	$2.2 * 10^{-10}$	$4.3 * 10^{-17}$

B. Error Bound of the Sum Cell

Benefiting from the separate recording of elephant and mouse flows and two conservative update strategies from the update and exchange process, the PNT sketch has a more rigorous error bound than the CM sketch and the Elastic sketch. We give specific analysis as follows.

Definition 5.1: Let $\|L_i\|_1$ denote the sum value of i-th array in the L part and L_{max} denote the maximum value in $\|L_1\|_1, \ldots, \|L_{d_L}\|_1$.

Theorem 5.1: Given two parameters ε and δ, let $w_L = \lceil \frac{e}{\varepsilon} \rceil$ (e is Euler number) and $d_L = \lceil \ln \frac{1}{\delta} \rceil$. The reported packet number $\hat{f}_i$ by the PNT sketch for flow i is bounded by

$$\hat{f}_i \leqslant f_i + \varepsilon \cdot L_{max} \tag{2}$$

with probability at least $1 - \delta$.

Proof 5.1: When a flow is recorded directly or exchanged into the H part, there is no error in subsequent updates, so we only need to calculate the error bound for the L part. We introduce indicator variable $I_{i,j,k}$, which is 1 if $(i \neq k) \wedge (h_j(i) = h_j(k))$, and 0 otherwise. By pairwise independence of the hash functions, then

$$E(I_{i,j,k}) = Pr[h_j(i) = h_j(k)] = \frac{1}{w_L} \leqslant \frac{\varepsilon}{e} \tag{3}$$

Define the variable $X_{i,j}$ to be $X_{i,j} = \sum_{k=1}^{n} I_{i,j,k} a_{j,k}$, where $a_{j,k}$ is the number of flow i packets that cause a counter in array j to add one. Subject to conservative update strategies, not all packets cause the counter to increase. So, $a_{j,k}$ is less than f_i, and varies across arrays. By linearity of expectation,

$$\begin{aligned} E(X_{i,j}) &= E(\sum_{k=1}^{n} I_{i,j,k} a_{j,k}) \\ &\leqslant \sum_{k=1}^{n} a_{j,k} E(I_{i,j,k}) \leqslant \frac{\varepsilon}{e} \|L_j\|_1 \end{aligned} \tag{4}$$

According to the definition, $L_j[h_j(i)\%w_L] = a_{j,i} + X_{i,j}$. By the Markov inequality,

$$\begin{aligned} & Pr[\hat{f}_i > f_i + \varepsilon \cdot L_{max}] \\ = & Pr[\forall_j . L_j[h_j(i)\%w_L] > f_i + \varepsilon \cdot L_{max}] \\ = & Pr[\forall_j . a_{j,i} + X_{i,j} > f_i + \varepsilon \cdot L_{max}] \\ \leqslant & Pr[\forall_j . X_{i,j} > \varepsilon \cdot \|L_j\|_1] \\ \leqslant & Pr[\forall_j . X_{i,j} > e \cdot E(X_{i,j})] \\ < & e^{-d} \leqslant \delta \end{aligned} \tag{5}$$

Therefore, theorem 5.1 holds. According to [7] and [3], when $w = w_L$ and $d = d_L$, the error bounds of the Elastic sketch and the CM sketch are $\hat{f}_i \leq f_i + \varepsilon \|f_l\|_1$ and $\hat{f}_i \leq f_i + \varepsilon \|f\|_1$, where $\|f_l\|_1$ represents the number of packets recorded in Elastic's light part and $\|f\|_1$ represents the number of all packets. In our experiments with the CAIDA trace, the error bound of the PNT sketch is about 1.2 and 3.7 times lower than that of the Elastic sketch and the CM sketch.

VI. Implementation

A. Software Implementation

To evaluate the performance of the PNT sketch, we reproduce four state-of-the-art algorithms (SuMax [11], CU [5], Elastic [7] and HashFlow [9]) based on the algorithms in the published papers, and implement them and the PNT sketch in Python on the x86 platform. For each algorithm, the default memory size is 0.5 MB. For Elastic, HashFlow and PNT, we set the size of each fid cell to 13 B, and the size of each sum cell in the light part, the ancillary table, and the L part to 8 bits. For other cells in all algorithms, we set the size to 32 bits. Except for HashFlow's auxiliary table, the d of each part, table, and sketch is set to 4.

B. P4 Implementation

To verify the feasibility of the PNT sketch in a production environment, we implement it on a P4 switch with $d_H = 2$ and $d_L = 2$. There are two main challenges: (1) the number of stages in a P4 switch is limited. A P4 program consists of multiple match-action tables and is compiled into a pipeline consisting of multiple stages. All dependent tables must be deployed in different stages. We carefully design our program and deploy over 1000 lines of P4 source code while meeting the stage limit; and (2) backtracking is not allowed in the pipeline. We use the $resubmit$ extern to deploy our exchange strategy, which requires revisiting some records. Because the number of exchange processes is only a small fraction ($< 1\%$ in our experiment) of the total process, the loss of throughput caused by the $resubmit$ extern can be ignored. Finally, we deploy our programs on a P4 switch and achieve line-speed packet processing.

VII. Evaluation

A. Experimental Setup

In this section, we evaluate the performance of the PNT sketch with the CAIDA trace specified in Section III.A. on seven different measurement tasks.

Measurement tasks:

- *Flow size estimation*: estimating the size of any flow. In this paper, we identify a particular flow with the 5-tuple (source IP address, destination IP address, source port, destination port, transport layer protocol) and take the number of packets as the flow size.
- *Heavy hitter detection*: reporting flows whose size is larger than a preset threshold. We set the threshold at $10 \sim 100$ and tested how different thresholds affect the algorithms' performance.
- *Heavy change detection*: reporting flows whose size in two adjacent measurement periods increases or decreases beyond a preset threshold. We set the threshold as λ of total changes over two adjacent periods, with λ of 0.01 $\sim$ 0.1.
- *Latest arrival time estimation*: estimating the lasted packet arrival time of each top-k flow.
- *Maximum IFPD estimation*: estimating the maximum IFPD of each top-k flow.
- *Average IFPD estimation*: estimating the average IFPD of each top-k flow.
- *High IFPD detection*: reporting flows whose maximum IFPD is larger than a preset threshold. We set the threshold at $100 \sim$ 1000ms.

By default, we set set the measurement period length to 10 s and the k to 3K, which is approximately 1% of the flow number of the CAIDA trace in a single period.

Evaluation metrics:

- *ARE (Average Relative Error)*: we use ARE to evaluate the accuracy of all estimation tasks. The formula is shown below, where n is the number of flows, and $\hat{f}_i$ and f_i are the estimated and actual flow attributes, respectively.

$$ARE = \frac{1}{n} \sum_{i=1}^{n} \frac{|\hat{f}_i - f_i|}{f_i}$$

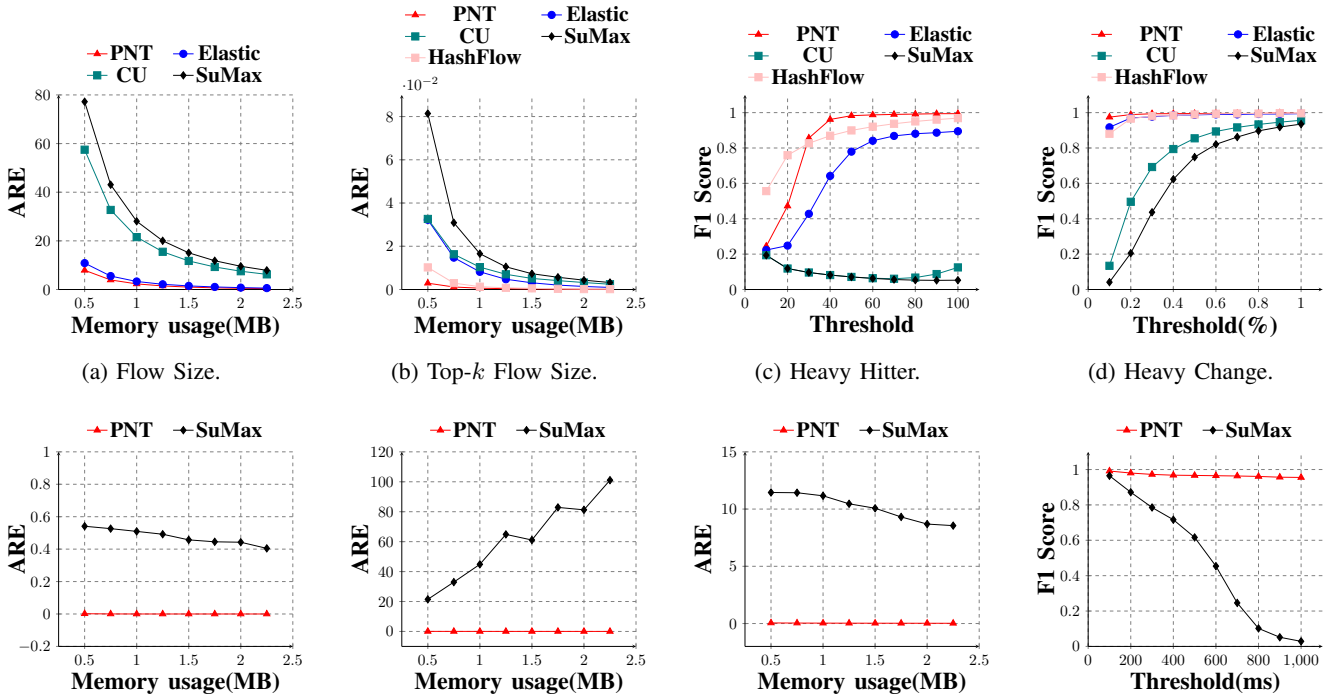
- *F1 Score*: we use the F1 score to evaluate the accuracy of all detection tasks. The formula is shown below, where N_{TP} is the number of true reported flows, N_R is the number of reported flows, and N_T is the number of true flows.

$$PR = \frac{N_{TP}}{N_R} \quad RR = \frac{N_{TP}}{N_T}$$

$$F1\ Score = \frac{2 * PR * RR}{PR + RR}$$

B. Accuracy

Flow size estimation (Fig. 6a-6b): We find that PNT consistently outperforms its competitors. When using 0.5MB of memory, for all flows, the ARE of PNT is about 9.8, 7.3, and 1.4 times lower than that of SuMax, CU, and Elastic. And for top-k flows, the ARE of PNT is about 27.9, 11.2, 11.0 and 3.5 times lower than that of SuMax, CU, Elastic, and HashFlow. It can be seen that PNT improves the accuracy of the top-k flows compared to other algorithms. Further analysis shows that, in the estimation of PNT, on average 82.1% of the top-k



(a) Flow Size. (b) Top-k Flow Size. (c) Heavy Hitter. (d) Heavy Change.

(e) Latest Arrival Time. (f) Maximum IFPD. (g) Average IFPD. (h) High IFPD.

Fig. 6: Accuracy comparison for seven tasks.

flows are error-free because of the history-aware reset strategy. They are recorded in the H part from the first packet, even if some of them start later. The ratios for Elastic and HashFlow are 24.6% and 72.9%, respectively.

Heavy hitter detection (Fig. 6c): When the threshold is tiny, HashFlow performs better because it only has under-estimation errors, thus avoiding many false positives caused by mouse flows and getting a higher precision. Other algorithms perform poorly due to the over-estimation of the size of mouse flows. The accuracy of PNT increases rapidly as the detection threshold becomes larger. When the threshold is set to 40, the F1 score of PNT is 0.96 and is about 11.76, 11.76, 1.50, and 1.11 times higher than that of CU, SuMax, Elastic, and HashFlow.

Heavy change detection (Fig. 6d): We find that the F1 score of PNT is higher than 0.97 for all threshold settings. The F1 scores of Elastic and HashFlow are close to but always lower than that of PNT.

Latest Arrival Time (Fig. 6e): We find that all top-k flows are recorded in the H part, causing the ARE of PNT to always be zero. The ARE of SuMax decreases as memory usage increases.

Maximum IFPD estimation (Fig. 6f): We find that even with only 0.5MB of memory, the ARE of PNT is still below 0.05. Errors are caused by the exchange process. Paradoxically, the ARE of SuMax increases as the memory increases. Our further analysis shows that this is because SuMax significantly overestimates the delay of the first packet for late arriving flows. Ideally, when updating, we should set the delay to zero for the first packet of each flow. But SuMax cannot accurately identify each first packet due to hash collisions, resulting in the wrong delay calculation as $t_{now} - t_{last}$. As the memory increases, the hash collisions decrease but still exist. Then t_{last} becomes relatively smaller, and the calculation errors become larger. As the memory continues to increase, the ARE of SuMax reaches a peak, which happens approximately when memory is 10 MB, and then decreases. At this point, there is no more hash collision for some flows.

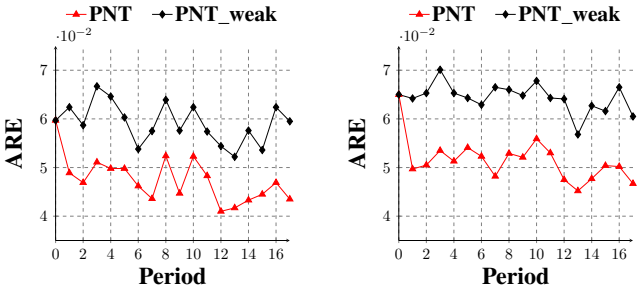
Average IFPD estimation (Fig. 6g): We find that the ARE of PNT is close to zero, and SuMax underperforms for the same reasons that it underperforms on the previous task. But errors of SuMax are less pronounced than on the previous task since estimating the average value.

High IFPD detection (Fig. 6h): We find that PNT achieves high accuracy at all threshold settings, and the F1 score is still higher than 0.95 in the worst case. In contrast, the F1 score of SuMax decreases as the threshold increases. This is due to the over-estimation of the maximum IFPD for most flows, which leads to a low precision when the threshold is large.

To summarize, we find that PNT can achieve a trade-off between high accuracy and low overhead. Using only 0.5MB of memory, PNT achieves very high accuracy on seven different measurement tasks. Especially for the top-k flows, PNT achieves accurate estimations, which we believe can be used for fine-grained network management, such as traffic engineering, congestion control, etc.

C. Accuracy vs History-aware Reset Strategy

In this section, we evaluate the performance of the history-aware reset strategy. We let PNT be reset at each period's



(a) Maximum IFPD. (b) Average IFPD.

Fig. 7: Performance of the history-aware reset strategy

start, naming it PNT_weak. Then, we use a 3-minute CAIDA trace and set the memory usage to 0.5 MB. We found that the strategy can significantly improve PNT accuracy on top-k related tasks. As shown in Fig. 7, at period 0, PNT and PNT_weak have the same performance because this is the first measurement period and there is no available historical record for PNT. In the following periods, the ARE of PNT is about 1.23 and 1.27 times lower than that of PNT_weak in the maximum and average IFPD estimation tasks, respectively. This result illustrates that PNT can effectively use the elephant flow records collected from the previous period to reserve space for new elephant flows.

D. Other Experiments

Effects of using the same hash functions: We let PNT use different hash functions in the H part and the L part, naming it PNT_DH. The experimental results show that for the estimation tasks, the ARE difference between PNT and PNT_DH is less than 4‰. And for the detection tasks, the F1 score of PNT is less than 2% lower than that of PNT_DH. However, the throughput of PNT is improved by an average of 63.8% compared to PNT_DH.

Performance of conservative update strategies: We let PNT use the update strategy of CM for the update and exchange processes, naming it PNT_NCU. The experimental results show that for all and top-k flow size estimations, the ARE of PNT is 1.35 and 1.53 times lower than that of PNT_NCU, respectively. Conservative update strategies act on the L part but can improve the accuracy of top-k flows. This is because they reduce the errors introduced by exchange processes. And the number of exchange processes dropped by 15.1%.

VIII. CONCLUSION

In this paper, we propose the PNT sketch, a generic sketch algorithm for periodic network telemetry. The three key techniques of our sketch are: (1) recording elephant and mouse flows separately and implementing multiple accurate aggregation functions to record flow attributes for elephant flows; (2) selecting appropriate parameters to record the vast majority of elephant flows in the h part with limited resources; and (3) identifying potential elephant flows in advance through a history-aware reset strategy to reserve space for them. Experimental results show that the PNT sketch achieves a trade-off between high accuracy and low overhead and outperforms the state-of-the-art for seven typical measurement tasks. In the future, we plan to design a more efficient history-aware reset strategy to further improve the accuracy of elephant flows and improve the P4 implementation of the PNT sketch.

REFERENCES

[1] Case, Jeffrey D., et al. Simple network management protocol (SNMP). No. rfc1098. 1989.
[2] Claise, Benoit. Cisco systems netflow services export version 9. No. rfc3954. 2004.
[3] Cormode, Graham, and Shan Muthukrishnan. "An improved data stream summary: the count-min sketch and its applications." Journal of Algorithms 55.1 (2005): 58-75.
[4] Charikar, Moses, Kevin Chen, and Martin Farach-Colton. "Finding frequent items in data streams." International Colloquium on Automata, Languages, and Programming. Springer, Berlin, Heidelberg, 2002.
[5] Estan, Cristian, and George Varghese. "New directions in traffic measurement and accounting." Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications. 2002.
[6] Deng, Fan, and Davood Rafiei. "New estimation algorithms for streaming data: Count-min can do more." Webdocs. Cs. Ualberta. Ca (2007).
[7] Yang, Tong, et al. "Elastic sketch: Adaptive and fast network-wide measurements." Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. 2018.
[8] Roy, Pratanu, Arijit Khan, and Gustavo Alonso. "Augmented sketch: Faster and more accurate stream processing." Proceedings of the 2016 International Conference on Management of Data. 2016.
[9] Zhao, Zongyi, et al. "Efficient and Accurate Flow Record Collection With HashFlow." IEEE Transactions on Parallel and Distributed Systems 33.5 (2021): 1069-1083.
[10] Zhao, Zongyi, et al. "Continuous Flow Measurement with SuperFlow." 2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS). IEEE, 2021.
[11] Zhao, Yikai, et al. "LightGuardian: A Full-Visibility, Lightweight, In-band Telemetry System Using Sketchlets." 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21). 2021.
[12] Mizrahi, Tal, et al. "AM-PM: Efficient network telemetry using alternate marking." IEEE Network 33.4 (2019): 155-161.
[13] CAIDA UCSD Anonymized Internet Traces Dataset - 2018, http://www.caida.org/data/passive/passive dataset.xml.
[14] Liu, Zaoxing, et al. "One sketch to rule them all: Rethinking network flow monitoring with univmon." Proceedings of the 2016 ACM SIGCOMM Conference. 2016.
[15] Sanjuàs-Cuxart, Josep, et al. "Sketching the delay: tracking temporally uncorrelated flow-level latencies." Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference. 2011.
[16] Sivaraman, Vibhaalakshmi, et al. "Heavy-hitter detection entirely in the data plane." Proceedings of the Symposium on SDN Research. 2017.
[17] Li, Yuliang, et al. "FlowRadar: A Better NetFlow for Data Centers." 13th USENIX symposium on networked systems design and implementation (NSDI 16). 2016.
[18] Zhu, Jiaqi, Kai Zhang, and Qun Huang. "A Sketch Algorithm to Monitor High Packet Delay in Network Traffic." (2021).
[19] Liu, Hongqiang Harry, et al. "Traffic engineering with forward fault correction." Proceedings of the 2014 ACM Conference on SIGCOMM. 2014.
[20] Bogle, Jeremy, et al. "TEAVAR: striking the right utilization-availability balance in WAN traffic engineering." Proceedings of the ACM Special Interest Group on Data Communication. 2019. 29-43.
[21] Estan, Cristian, and George Varghese. "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice." ACM Transactions on Computer Systems (TOCS) 21.3 (2003): 270-313.
[22] Grabisch, Michel, et al. Aggregation functions. Vol. 127. Cambridge University Press, 2009.