The Power of Slicing in Internet Flow Measurement

Ramana Rao Kompella
University of California, San Diego
ramana@cs.ucsd.edu

Cristian Estan
University of Wisconsin-Madison
estan@cs.wisc.edu

Abstract

Flow measurement evolved into the primary method for measuring the composition of Internet traffic. Large ISPs and small networks use it to track dominant applications, dominant users, and traffic matrices. Cisco's NetFlow is a widely deployed flow measurement solution that uses a configurable static sampling rate to control processor and memory usage on the router and the amount of reporting traffic generated. Proposed enhancements to the basic Sampled NetFlow solve some of its problems. Smart Sampling reduces the overhead of reporting and storing the flow records generated by NetFlow by sampling them with probability proportional to their byte counts. Adaptive NetFlow limits memory and CPU consumption at the router by dynamically adapting the sampling rate used by NetFlow.

In this paper we propose "Flow Slices", a flow measurement solution that can be deployed through a software update at routers and traffic analysis workstations. Flow Slices, inspired from Smart Sampling and Adaptive NetFlow, introduces novel ideas such as – separating sampling rate adaptation from measurement bins; controlling the three resource bottlenecks at the router (CPU, memory, reporting bandwidth) using separate "tuning knobs"; basing smart sampling decisions on multiple factors; a flow measurement algorithm related to sample and hold; new estimators for the number of bytes and flows. The resulting solution has smaller resource requirements than current proposals and it enables more accurate traffic analysis results. We provide theoretical analyses of the unbiasedness and variances of the estimators based on Flow Slices and experimental comparisons with other flow measurement solutions such as Adaptive NetFlow.

1 Introduction

The role of traffic measurement in operating large scale IP networks requires little or no introduction. Traffic measurement allows network operators to make informed decisions about provisioning and extending their networks, and it helps solve many operational problems. Specialized devices operating on relatively low traffic links can perform complex security analyses that reveal malicious activities [18, 20], monitor complex performance metrics [6], or simply capture packet (header) traces with accurate timestamps [7] to be analyzed offline. Much simpler solutions such as SNMP counters [16] are deployed on even the highest speed links, but they only give measurements of the total volume of the traffic. Flow level measurement at routers [2, 3] offers a good compromise between scalability and the complexity of the traffic analyses supported since it can offer details about the composition of the traffic mix.

In this paper, we propose a new flow measurement solution: *Flow Slices*. The contributions of this paper are both practical and theoretical and we summarize the most important ones here.

- Flow Slices has separate parameters controlling the three possible bottlenecks at the router: processing load, memory, and reporting bandwidth. This makes it easier to fit this solution into various implementation scenarios.

- The flow slicing algorithm at the core of this solution provides more accurate results than packet sampling using the same amount of memory and it enables new measures of traffic such as estimates for the number of active flows.

- Flow Slices separates sampling rate adaptation from binning and thus provide a solution with the robustness of Adaptive NetFlow without paying the extra cost in memory and measurement bandwidth due to binned measurement. See Table 1 for a comparison of various flow measurement solutions.

- We propose multifactor smart sampling that takes into account multiple factors such as byte counts, packet counts, and the existence of SYN flags in the flow record to determine the sampling probability for individual flow records. For comparable configurations this decreases significantly the variance in estimates of the number of flow arrivals while increasing only slightly the variance for byte counts when compared to Smart Sampling.

- Optional binned measurement allows us to eliminate binning error in the analysis phase, while still maintaining the memory and reporting bandwidth overheads below those of Adaptive NetFlow.

- We propose novel estimators $\widehat{b}$, $\widehat{f}$, $\widehat{A}^{(1)}$, and $\widehat{A}^{(2)}$ for various measures of traffic. See Section 4 for a discussion of these and other estimators.

Before we explain Flow Slices, we briefly review some of the previous work in this area of Internet flow measurement.

Issue	Sampled NetFlow	Adaptive NetFlow	Flow Slices
Memory usage	Variable	Fixed	Fixed
Volume of flow data reported	Variable	Fixed	Fixed
Behavior under DDoS with spoofed sources and other traffic mixes with many flows	Panicky flow expiration	Reduction in accuracy	Small reduction in accuracy
Estimates of traffic in small time bins	Less accurate	Accurate	Less accurate
Reporting overhead when using small bins	Unaffected	Large increase	Unaffected
Lifetime of flow record in router memory	Min(active timeout, flow length + inactive timeout)	Bin length	Min(slice length, flow length + inactive timeout)
Resource usage at end of time bin	N/A	Reporting spike or extra memory	N/A
Processing intensive tasks	Counting	Counting and renormalization	Counting
Counting TCP flow arrivals (using SYNs)	Yes	Yes	Yes
Counting all active flows	No	Separate flow counting extension	Yes
Counting all active flows at high speeds	No	Hardware flow counting extension	No

Table 1: Sampled NetFlow, Adaptive NetFlow and Flow Slices differ in the types of measurements they support, in how they adapt to different traffic mixes, and in their resource consumption (memory usage and reporting traffic).

2 Related work

NetFlow [17], first implemented in Cisco routers, is the most widely used flow measurement solution today. Routers maintain flow records collecting various bits of information. Flows are identified by fields present in the header of every packet[1]: source and destination IP address, protocol, source and destination port, and type of service bits. The flow record keeps information such as the number of packets in the flow, the (total) number of bytes in those packets, the timestamp of the first and last packet, and protocol flag information such as whether any of those packets had the SYN flag set. NetFlow uses four rules to decide when to remove a flow record from router memory and report it to the collection station: 1) when TCP flags (FIN or RST) indicate flow termination, 2) 15 seconds (configurable “inactive timeout”) after seeing the last packet with a matching flow ID, 3) 30 minutes (configurable “active timeout”)’ after the record was created to avoid staleness and 4) when the memory is full.

On every new packet, NetFlow looks up the corresponding entry (creating a new entry if necessary) and updates that entry’s counters and timestamps. Since for high speed interfaces, the processor and the memory holding the flow records cannot keep up with the packet rate, Cisco introduced Sampled NetFlow [22] which updates the flow cache only for sampled packets. For a configurable value of a parameter N, a packet is sampled with one in N probability.

One problem with NetFlow is that the memory required by the flow records and the bandwidth consumed to report them depends strongly on the traffic mix. In particular, large floods of small packets with randomly spoofed source addresses can increase memory and bandwidth requirements by orders of magnitude. Adaptive NetFlow [10] solves this problem by dynamically adapting the sampling rate. Adaptive NetFlow divides the operation of the flow measurement algorithm into equally spaced time bins.[2] Within each bin, the algorithm starts by sampling aggressively (high sampling probability). If memory is consumed too quickly, it switches to less aggressive sampling. It then “renormalizes” existing entries so that they reflect the counts they would have had with the new sampling rate in effect from the beginning of the bin. At the end of the bin, all entries are reported.

Using fixed size bins in Adaptive NetFlow increases the memory utilization and can cause bursty spikes in bandwidth consumption as compared to Sampled NetFlow. Memory utilization is higher because, to operate seamlessly between bin-boundaries, Adaptive NetFlow requires two sets of records (double-buffering), one for current bin and one for records in the previous bin while they are being transmitted. Without double-buffering, flow records that expire at the bin-boundary need to be transmitted immediately at very high bandwidth in order to create space for the next set of entries.

[1] Technically the incoming router interface is also part of the flow identifier.

[2] Typically, traffic statistics are analysed in time bins, and hence, bin sizes are chosen based on the granularity of traffic statistics.

Large flows spanning multiple bins will be reported separately for every bin increasing the bandwidth consumption. Table 1 gives a summary comparison of Sampled NetFlow, Adaptive NetFlow and Flow Slices.

These flow records are then used to estimate the number of bytes or packets in various traffic aggregates of interest. This can give network operators information about dominant applications, the network usage of various clients, traffic matrices, and many other useful statistics [12, 19, 1, 14]. Smart Sampling [8] is a way of reducing the data used by such analyses without significantly affecting their results. Smart Sampling retains flow records with probability proportional to the size of their byte counter. The flow records can also be used to estimate the number of active flows which is important when looking for denial of service attacks, scans, and worms in the traffic mix. Unfortunately, if we use Sampled NetFlow it is impossible to recover the number of flows in the original traffic from the collected data [5] unless we use protocol information. By looking at the SYN flag information in flow records it is possible to accurately estimate the number of TCP flows in the traffic mix [9].

3 Description of flow slices

The core flow slicing algorithm is based on the sample and hold algorithm [11]. After presenting the core algorithm, we discuss four extensions: adding packet sampling to scale to high speed links, using an inactivity timeout to reduce memory usage at router, adding binned measurement to reduce binning error during analysis, and adding a variant of smart sampling to control the volume of flow data reported. The version of Flow Slices described used for Table 1 has the first two extensions. We also discuss the configuration parameters of the complete flow slicing solution and how they can be set adaptively based on the current traffic mix.

3.1 Core algorithm

The core flow slicing algorithm addresses the problem of reducing the memory usage of the flow measurement module. Sampled NetFlow and Adaptive NetFlow use random packet sampling: they only handle sampled packets. Just as sample and hold [11], flow slicing uses sampling only to control the creation of flow entries, once a sampled packet creates an entry for a flow, all its subsequent packets are counted (not just the sampled ones). This increases the accuracy of the estimates of packet counts, without changing the memory requirement. We use the "flow slicing probability" p to control the creation of flow entries. We expire and report each entry exactly t seconds after its creation, irrespective of the rate at which packets arrive for a particular flow. [3]. Just as in the case of NetFlow, the entry associated with a flow has a byte and packet counter updated at every packet, timestamps for the first and last packet, and it stores protocol information such as whether any of the packets counted against the entry had the SYN flag set. To ensure unbiasedness of estimators, on creation of an entry we do not initialize the byte counter to the number of bytes b_{first} in the packet that caused the creation of the entry, but to b_{first}/p (see Section 4.2 for more details).

The slice length t is related to the "active timeout" of NetFlow which controls for how long an active entry is kept before expiring and being reported (default 30 minutes). Both of these parameters limit the staleness of the data (i.e. if we have a long-lived flow, we know that its traffic will be reported with at most this much delay).

By dynamically adapting the flow slicing probability, we can control the rate at which entries are created and freed, thus ensuring that the algorithm stays within its allocated memory budget M. By keeping the rate at which entries are created, on average slightly below M/t, we can also keep the rate at which flows records are reported smooth. In contrast Adaptive NetFlow proposes expiring all active entries at the end of the measurement bin, so it either has a large peak in reports, or it requires buffers that increase the memory usage by almost a factor of two if the reporting of the records is smoothed out over the next measurement bin. We do not however, discuss dynamic adaptation in much detail in this paper, as adaptation techniques similar to that in [10] can be applied in this context using feedback from the current memory usage. Note however, that in our adaptation, we do not require the costly operation of renormalization that is required in Adaptive NetFlow. Next we discuss some of the tuning knobs we provide to control the three resource bottlenecks (CPU, Memory, Bandwidth).

3.2 Scaling to high speeds

The flow slicing probability p controls the memory usage, but since we do a lookup in the flow memory for every packet, flow slicing does not control the processing load. In the presence of limited processing power, we add a random packet sampling stage in front of the flow slicing stage (see Figure 1). A simple solution is to set the packet sampling probability q statically to a value that ensures that the processor performing the flow measurement can keep up even with worst case traffic mixes. Based on Cisco recommendations [17] for turning on NetFlow sampling for speeds higher than OC-3,

[3] We call our proposal "flow slices" because each entry tracks a "slice" of length t from the flow. We could extend the flow slice termination condition to protocol specific hints such as FIN or RST flags, but since these are not reliable in the presence of packet sampling which we use as a first stage, we ignore them in this paper.

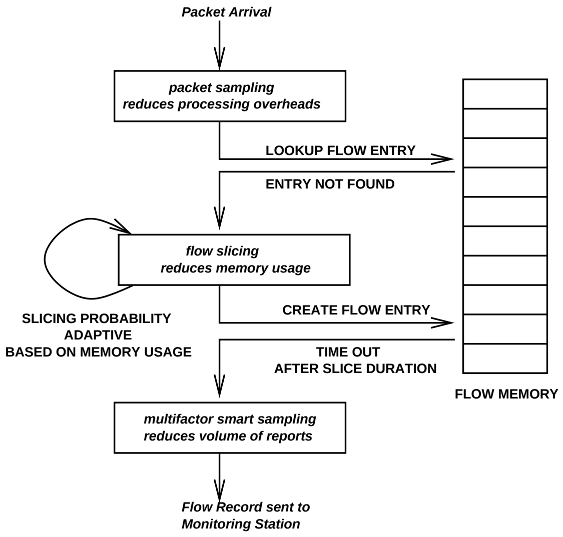

Figure 1: Architecture

we set q to $1/4$ for OC-12 links, $1/16$ for OC-48, etc. With these packet sampling rates, and with worst case traffic consisting of the link entirely full with 40 byte packets, the flow measurement module has more than 0.5μ per packet and it has time to perform between 8 and 9 (wide) DRAM accesses on average.

3.3 Adding an inactivity timer

Most flows in the Internet are short-lived. If our only mechanism for removing an entry is its expiration after the slice length t and we use a large value for t, at any moment in time, most of the entries in the flow memory will belong to flows that are no longer active and just use up memory waiting to expire. On the other hand having a very short slice length can lead to an increase in reporting traffic and loss of accuracy. Adding an inactive timeout parameter $t_{inactive}$ to flow slices reduces the memory spent on obsolete entries. Experimental results in Section 6.1 show that we could significantly reduce the memory requirement if we deploy inactivity timers. An adaptive algorithm for setting the flow slicing rate can turn this reduction in memory usage into an increase in accuracy.

3.4 Adding binned measurement

With flow slices we have the same problem as with NetFlow if we want to perform traffic analysis using time bins: for flow slices that span time bins, we can only guess how many of the flow's packets were in each bin, and this introduces errors in the results. This problem is even more pronounced when analysis is required in very small time bins to capture more precise traffic dynamics. We can extend flow slices to support binned measurement of traffic by keeping multiple sets of byte and packet counters, one set for each bin the slice passes through. By keeping separate counters for each bin, the binning error is eliminated entirely, at the cost of increasing the size of the flow records. Note that the reporting bandwidth costs of this solution are significantly smaller than those of the solution used by Adaptive NetFlow where an entire record is reported for each bin. The byte and packet counters are 8 bytes whereas a complete record is 48 bytes.

The number of counters per record has to be one larger than the number of bins required to fit a slice because the flow slice can overlap only partially with the first and last bin. The choice of the size of the measurement bin supported is a compromise between resource consumption at the router and accuracy of results. Reasonable choices can range anywhere from the slice length t to 20 times smaller. For brevity, we do not explore this further in the paper, but note that depending on the final goal, flow slicing algorithm can be extended with additional resources to obtain desired accuracy.

3.5 Controlling the reporting bandwidth

Smart sampling has been proposed as a way of reducing the number of flow records without causing much error. Smart sampling focuses on measuring the number of bytes in arbitrary aggregates of traffic and thus smart sampling favors flow records with large byte counters over those with small flow counters. Common packet sizes vary between 40 and 1500, so while the packet counts are not proportional to the byte counts, they are closely correlated. Thus smart sampling will ensure that the errors introduced in packet counts are also small. The situation is different with flow arrival counts. These depend heavily on flow records with the SYN flag set, and most such records come from small flows which are discriminated against by smart sampling. Thus the errors introduced by smart sampling in the flow arrival counts are significant.

We propose a new variant of smart sampling, *multifactor smart sampling* which takes into consideration not just byte counts, but also packet counts and SYN flags. While multifactor smart sampling still favors flow records with large byte and packet counts, it also favors records with the SYN flag, thus ensuring that the errors introduced into the flow arrival counts are not large either. Because the exact rule used to determine the multifactor smart sampling probability r depends on estimators of byte and packet counts, we postpone its discussion to Section 4.5.

3.6 Setting the parameters of flow slicing

Routers or other network devices performing flow measurement have three types of resources that can become bottlenecks: processing power, flow memory, and re-

Parameter	What it controls	How it is set
Flow slicing probability	Memory usage at router	Adaptively based on memory usage
Flow slice length	Staleness of reported data	Statically based on user preferences
Inactivity timeout	Reduces memory usage	Statically based on typical inter packet arrival time
Packet sampling probability	Processing load at router	Statically based on worst case traffic
Bin size (optional)	Binning error	Statically based on user preferences
Smart sampling thresholds	Volume of flow data reported	Statically or adaptively based on target volume

Table 2: Configuration parameters for Flow Slices.

porting bandwidth. Flow slices use three different "tuning knobs" to control these three resources: the packet sampling probability q controls the processing load, the flow slicing probability p controls the memory usage and the thresholds determining the smart sampling probability r control the volume of data reported. This can result in more accurate traffic analysis results than using a single parameter, the packet sampling probability, to control all three resources, as Adaptive NetFlow does. This distinction would be irrelevant in practice if the only scarce resource would be the processing power at the router, so it is useful to perform a quick sanity check before proceeding any further: can an unfavorable traffic mix push the memory requirements or reporting bandwidth so high that they become a problem? Let's first assume a traffic mix consisting of back to back minimum sized packets, each belonging to a different flow (a massive flooding attack with randomly spoofed source addresses). With the packet sampling rates from Section 3.2, the traffic measurement module would receive a packet every 0.5μ. Even with an aggressive inactive timeout of $t_{inactive} = 5$ seconds, we need a flow memory that can fit $10,000,000$ flow records, which at 64 bytes/record[17] requires 610 megabytes. When reported flow records take 48 bytes (ignoring overheads), so at $2,000,000$ flow records/second, which requires 768 megabits/second. These numbers are orders of magnitude above what one can comfortably afford. The experiments from Section 6 use realistic traffic mixes to evaluate the benefits of Flow Slices as compared to Sampled NetFlow and Adaptive NetFlow as opposed to pathological traffic scenarios.

For each of the parameters of Flow Slices listed in Table 2 we need to decide whether to set them statically as part of the router configuration, or dynamically adapt them to the current traffic mix. Of the three main tuning knobs, the flow slicing probability p should definitely be set dynamically to allow the router to protect from memory overflow when faced with unfavorable traffic mixes. The thresholds controlling the smart sampling probability can also be set adaptively. In this paper we consider that the packet sampling probability q is static based on recommended values for different link capacities. Flow Slices would work just as well with a dynamic packet sampling probability that could go above the conservative static value, but since it is hard to guarantee the stability of such approach without pushing the packet sampling rate adaptation logic into hardware (which raises deployment problems), we chose not to explore such a solution in this paper.

The observant reader might have noticed that without the optional binned measurement feature Flow Slices resemble Sampled NetFlow. If the dynamic adaptation algorithms set the flow slicing probability p and the smart sampling probability r to 1 the two solutions perform exactly the same processing. We consider this to be an important feature. The difference between Sampled NetFlow and Flow Slices is in how they react to unfriendly traffic mixes and environments with strong constraints on resources. While both Adaptive NetFlow and Flow Slices provide robustness to unfavorable traffic mixes, Adaptive NetFlow forces the user to adopt the binned measurement model (which can increase memory usage and the volume of reports) even when the traffic mix is favorable.

4 Estimators based on flow slices

In this section we discuss formulas for estimating traffic based on the flow records provided by Flow Slices. In practice, the user would be interested in the number of bytes, packets or flows in the entire traffic mix or a portion of it (e.g. the HTTP traffic, the traffic coming from a certain customer, etc.). All our estimators focus on a single flow. To compute the total traffic the user has to sum the contributions of all individual flow records. If the estimators for individual flows have the property of unbiasedness, the errors in the estimates for individual flows will not accumulate, but cancel out (to some extent). This is the reason why, in this section, we not only discuss the various estimators, but also show that they are unbiased.

For the purposes of our analysis, a bin is an arbitrary interval of time. This is not to be confused with the traffic analysis bins or Adaptive NetFlow's definition of bin. We will start by focusing on the simple case of a single bin, with slice length t and inactive timeout $t_{inactive}$

Name	Meaning
p	flow slicing probability
q	packet sampling probability
r	smart sampling probability
s	size of flow (in packets) before flow slicing
c_s	packet counter in flow record
$\widehat{s}$	estimate of the size of flow before flow slicing (0 if flow not sliced)
S	original size of flow (in packets) before packet sampling
$\widehat{S}$	estimate of the original size of flow (0 if flow not sampled or not sliced)
b	size of a flow in bytes before flow slicing
c_b	byte counter in flow record
$\widehat{b}$	estimate of the number of bytes in flow based on flow slices (0 if flow not sliced)
B	original size of flow in bytes before packet sampling
$\widehat{B}$	estimate of the original size of flow in bytes (0 if flow not sampled or not sliced)
$\widehat{f}$	contribution to the estimate of the number of active flows (0 if flow not sliced)
$\widehat{a}$	contribution to the estimate of the number of flow arrivals (0 if flow not sliced)
$\widehat{A}^{(1)}$	contribution to first estimator of number of flow arrivals (0 if flow not sampled or not sliced)
$\widehat{A}^{(2)}$	contribution to second estimator of number of flow arrivals (0 if flow not sampled or not sliced)
z_s	smart sampling threshold controlling the influence of $\widehat{S}$ on r
z_b	smart sampling threshold controlling the influence of $\widehat{B}$ on r
z_a	smart sampling threshold controlling the influence of $\widehat{A}^{(1)}$ on r

Table 3: Notation used in this paper.

larger than the size of the bin and flow memory empty at the beginning of the bin.

Next we will look at how the estimators generalize when we remove these constraints. Table 3 summarizes notation used throughout the paper.

4.1 Estimating packet counts

The packet counter c_s in an entry is initialized to 1 when the first packet of the flow gets sampled, and it is incremented for all subsequent packets belonging to the flow. Let s be the number of packets in the flow at the input of the flow slicing algorithm. Equation 1 gives the formula for our estimator $\widehat{s}$ for the number of packets in the flow.

$$\widehat{s} = 1/p - 1 + c_s \tag{1}$$

Lemma 1 *$\widehat{s}$ as defined in Equation 1 is an unbiased estimator of s.*

Proof: By induction on the number of packets s.

Base case: If $s = 1$, the only packet of the flow is sampled with probability p and in that case it is counted as $1/p-1+1 = 1/p$ packets. With probability $1-p$ it is not sampled (and it counts as 0). Thus $E[\widehat{s}] = p \cdot 1/p + 0 = 1 = s$.

Inductive step: By induction hypothesis, we know that for a flow with $s' = s-1$, $E[\widehat{s'}] = s' = s-1$. Also since the flow slice length t and the inactive timeout $t_{inactive}$ are larger than the bin size, we know that once the flow gets an entry, all its packets within the bin will get counted by c_s. There are two possible cases: the first packet of the flow gets sampled, and we get $c_s = s$, or it doesn't and than the value of c_s and $\widehat{s}$ will be the same as those for a flow with $s' = s-1$ packets for which the sampling decisions are the same as for the rest of the packets of our flow.

$$\begin{aligned} E[\widehat{s}] &= p \cdot (1/p - 1 + s) + (1-p)E[\widehat{s'}] \\ &= 1 - p + ps + (1-p)(s-1) = s \end{aligned}$$

■

If we sample packets randomly with probability q before applying the flow slicing algorithm, we will want to estimate the number of packets S at the input of the packet sampling stage. Since $E[s] = qS$, it is easy to show that $\widehat{S} = 1/q\widehat{s}$ is an unbiased estimator for S.

4.2 Estimating byte counts

Before discussing, the solution adopted by flow slices to estimate the number of bytes in a flow, we show why a simpler solution does not work. We could have the byte counter c_b in the flow entry just count the total number of bytes in the packets seen once the flow record is created. Just like with the packet counter, we need an additive correction to account for the packets missed before the creation of the entry. We can get an unbiased estimate for the number of packets missed, but not for their total size, because we do not know their sizes. We could assume that the packet sizes are uniform within the flow, but this would lead to systematic biases because they are not. As the proof of Lemma 2 shows, storing the size of the packet that was sampled and caused the creation of the entry would solve the problem because using it to estimate the total number of bytes in the packets not counted does lead to an unbiased estimator. But this would require another entry in the flow record. We decided instead to store this information in the byte counter itself by **initializing** c_b **to** b_{first}/p when the entry is created (b_{first} is the size in bytes of the sampled packet). Let b be the number of bytes of the flow at the input of the flow slicing algorithm.

$$\widehat{b} = c_b \tag{2}$$

Lemma 2 $\widehat{b}$ *as defined in Equation 2 is an unbiased estimator of* b*.*

Proof: By induction on the number of packets in the flow s. Let b_i for i from 1 to s be the sizes of the individual packets. By definition the number of bytes in the flow is $b = \sum_{i=1}^{s} b_i$. For convenience of notation we index the packet sizes in reverse order, so b_1 will be the size of the last packet and b_s the size of the first one.

Base case If s=1, the only packet is sampled with probability p and in that case it is counted $c_b = b_1/p = b/p$ bytes. With probability $1 - p$ it is not sampled (and it counts as 0). Thus $E[c_b] = p \cdot b/p + 0 = b$.

Inductive step By induction hypothesis we know that if the first packet is not sampled we are left with the last $s' = s - 1$ packets and $E[c_b] = b' = b - b_s$. If the first packet gets sampled, we count it as b_s/p and we count the rest exactly because the flow slice length t and the inactive timeout $t_{inactive}$ are larger than the bin size.

$$\begin{aligned} E[c_b] &= p \cdot (b_s/p + b') + (1-p)b' \\ &= b_s + pb' + (1-p)b' = b_s + b' = b \end{aligned}$$

■

If we sample packets randomly with probability q before applying the flow slicing algorithm, we will want to estimate the number of bytes B at the input of the packet sampling stage. Since $E[b] = qB$, it is easy to show that $\widehat{B} = 1/q\widehat{b}$ is an unbiased estimator for B.

4.3 Estimating the number of active flows

We use two definitions for counting flows: active flows and flow arrivals. A flow is active during a time bin if it sends at least one packet during that time bin. Multiple TCP connections that happen to share the same port numbers are considered a single flow and they will be reported in the same flow record under our current assumptions[4]. Active flows with none of their packets sampled by the flow slicing process will have no records so at least some of the flow records we get we should count as more than one active flow so that the total estimate will be unbiased. Our rule is to count records with a packet counter c_s of 1 as $1/p$ flows and other records as 1 flow and this gives us unbiased estimates for the number of active flows.

$$\widehat{f} = \begin{cases} 1/p & \text{if } c_s = 1 \\ 1 & \text{if } c_s > 1 \end{cases} \tag{3}$$

Lemma 3 $\widehat{f}$ *as defined in Equation 4 has expectation* 1.

Proof: There are three possible cases: if a packet before the last gets sampled, $c_s > 1$, if the last packet gets sampled $c_s = 1$, and if none of the packets gets sampled there will be no flow record, so the contribution of the flow to the estimate of the number of active flows will be $\widehat{f} = 0$. The probability of the first case is $p_{s-1} = 1 - (1-p)^{s-1}$, the probability of the second is $p(1 - p_{s-1})$ and that of the third is $(1-p)(1-p_{s-1})$.

$$\begin{aligned} E[\widehat{f}] &= p_{s-1} \cdot 1 + p(1-p_{s-1}) \cdot 1/p + \\ &\quad (1-p)(1-p_{s-1}) \cdot 0 = 1 \end{aligned}$$

■

The estimators for the number of bytes and packets in a flow were trivial to generalize to the case where we apply random packet sampling before flow slicing because the expected number of packets and bytes after packet sampling was exactly q times the number before. For the number of active flows there is no such simple relationship and actually it has been shown that it is impossible to estimate without significant bias the number of active flows once random sampling has been applied [5]. But by changing slightly the definition of flow counts we can take advantage of the SYN flags used by TCP flows.

4.4 Estimating flow arrivals

Flow arrivals are defined only for TCP flows which should start with one SYN packet. A flow is considered to have arrived in a bin if its SYN packet is in that time bin. Flows active during a certain bin, but with their SYN packet before the bin do not count as flow arrivals for that bin (but they count as active flows). If we look a the core flow slicing algorithm we can use the following estimator to compute the number of flow arrivals.

$$\widehat{f} = \begin{cases} 1/p & \text{if SYN flag set} \\ 0 & \text{if SYN flag not set} \end{cases} \tag{4}$$

[4] This way of defining flow counts is equivalent to an SQL query doing "COUNT DISTINCT" on flow identifiers seen during the time bin.

Given that the SYN flag is set in the flow record if it was set in *any* of the packets counted against the record, it is trivial to prove that $\widehat{f}$ leads to unbiased estimates of the number of flow arrivals if we make an assumption.

Assumption 1 Only the first packet for the flow can have the SYN flag set.

The flow arrival information is preserved by random packet sampling. Duffield et al. propose two estimators of the number of flow arrivals that work based on flow records collected after random sampling of the traffic [9]. The formulas for the individual contributions of flow records to the total estimate of the number of flow arrivals are as follows.

$$\widehat{M}^{(1)} = \begin{cases} 1/q & \text{if SYN flag set} \\ 0 & \text{if SYN flag not set} \end{cases}$$

$$\widehat{M}^{(2)} = \begin{cases} 1/q & \text{if SYN flag set and } s = 1 \\ 1 & \text{if SYN flag not set or } s > 1 \end{cases}$$

Duffield et al. show [9] that both estimators are unbiased $E[\widehat{M}^{(1)}] = E[\widehat{M}^{(2)}] = 1$ for flows that have exactly one SYN packet (which is implied by assumption 1). Both estimators overestimate the number of flow arrivals if flows have more than 1 SYN packet. For flows without any SYN packets which according to our definition of flow arrivals[5] should not be counted, we have $E[\widehat{M}^{(1)}] = 0$ and $E[\widehat{M}^{(2)}] > 0$, so to make the second estimator unbiased we need another assumption.

Assumption 2 The first packet within the bin for every flow has the SYN flag set.

Since the flows retaining SYN packets after the random packet sampling stage will retain a single SYN packet, and $\widehat{M}^{(1)}$ estimates the number of flow arrivals based on the number of such flows, we can easily combine it with $\widehat{a}$ to obtain an estimator for the number flows arrivals for the combined algorithm that does random packet sampling and flow slicing.

$$\widehat{A}^{(1)} = \begin{cases} 1/(pq) & \text{if SYN flag set} \\ 0 & \text{if SYN flag not set} \end{cases} \quad (5)$$

$\widehat{M}^{(2)}$ treats separately flows that only have a SYN packet after packet sampling and the others that survive it. Fortunately we can differentiate between the two types of flows even after flow slicing is applied: if a flow with a single SYN packet is sampled by flow slicing its record will have $c_s = 1$ and the SYN flag set; if any other flow is sampled by flow slicing and it has $c_s = 1$ it means that only its last packet was sampled thus it will not have the SYN flag set because that would put it into the category of flows with a single SYN packet surviving the packet sampling. Thus we can combine $\widehat{M}^{(2)}$ with $\widehat{a}$ to obtain another estimator.

$$\widehat{A}^{(2)} = \begin{cases} 1/(pq) & \text{if SYN flag set and } c_s = 1 \\ 1/p & \text{if SYN flag not set and } c_s = 1 \\ 1 & \text{if SYN flag not set and } c_s > 1 \end{cases} \quad (6)$$

Note that if assumption 1 is violated and we have more than one SYN packet at the beginning of the flow, say due to SYN retransmissions, both estimators will be biased towards overcounting. But if repeated SYNs are a rare enough occurrence, the effect on a final estimate based on many flow records will be small.

4.5 Multifactor smart sampling

To reduce the number of flow records, while maintaining accurate byte counts, smart sampling [8] proposes sampling the flow records with a size dependent probability $r = \min(1, b/z)$ where z is a threshold parameter controlling the tradeoff between the loss in accuracy and the reduction in the volume of reports. We can adapt smart sampling to flow slices using $r = \min(1, \widehat{B}/z)$ and we could still estimate byte, packet and flow arrival counts based on the smart sampled flow records using $\widehat{\mathcal{S}} = 1/r\widehat{S}, \widehat{\mathcal{B}} = 1/r\widehat{B}$, and $\widehat{\mathcal{A}} = 1/r\widehat{A}^{(1)}$. But using this formula for r results in a variance for $\widehat{\mathcal{A}}$ much larger than that of $\widehat{A}^{(1)}$ because it discriminates against flows with few bytes, and since most flows have few bytes, they will also produce most flow records with the SYN flag set – and these are exactly the records $\widehat{A}^{(1)}$ relies on.

We propose a new variant of smart sampling, multifactor smart sampling which takes into consideration not just byte counts, but also packet counts and SYN flags. By picking a smart sampling probability of $r = \min(1, \widehat{s}/z_s + \widehat{B}/z_b + \widehat{A}/z_a)$ we can balance the requirements of the three estimators. The three individual thresholds control the tradeoff between accuracy and reduction in report volume separately for the three estimators of bytes, packets and flow arrivals.

4.6 Dynamically adjusting the flow slicing probability

Flow Slices dynamically adjusts the flow slicing probability p to the current traffic. This adjustment can happen in the middle of a time bin. Which one of the many values of p should we use in our estimators? Are the estimators still unbiased? Actually none of the proofs depends on having a single value for p, and they would all work if we replaced it with a separate p_i for every packet. All the estimators would need to use the value of the packet slicing probability in effect at the time the sampling of a packet caused the creation of the entry. This doesn't necessarily mean that one needs to extend the flow entry with one more field, because it already holds the timestamp of the first packet and that can be used to determine

[5] Our definition of flow arrivals differs from that used in [9].

the flow slicing rate if the router keeps a small log of recent adjustments to it.

When the flow record expires and it is reported, the report should include the value of the flow slicing probability p in effect at the time the entry was created. Similarly if the smart sampling thresholds z_s, z_b, and z_a are adjusted dynamically, the report should include their current value so that one can compute r during analysis. But reporting all these parameters doesn't require an increase in the flow record size. For example they can be reported just once in every report packet if their value is the same for all the records reported together.

4.7 Bins, timeouts, and flow reconstruction

To simplify our discussion of the estimators we started with some strong assumptions: all records last longer than the bin length, counters count only packets within the bin of interest, and the flow memory is empty at the beginning of the bin. In this section we relax these assumptions and discuss the effects of these relaxations on the estimators.

4.7.1 Continuous operation

The most elementary relaxation of the assumption is to consider continuous operation of the algorithm: records still last longer than the bin length, and we still have separate counters for each bin, but there can be active records at the start of our bin, records created earlier.

The simplest case is that of records spanning the entire bin. The byte and packet counters will reflect the actual traffic, so we use $\widehat{S} = 1/qc_s$ and $\widehat{B} = 1/qc_b$. If we do not have a packet sampling stage we can also compute $\widehat{f} = 1$ if $c_s > 0$ and $\widehat{f} = 0$ otherwise. $\widehat{A} = 0$ because the flow started in an earlier bin.

If a flow record expires within the bin we run the analysis on, it can be the only record for the flow, but it is also possible that another record for the same flow would get created after the first record's expiration. For byte and packet counts which are additive we can just add the counters from the first record to the estimates from the second $\widehat{s} = \widehat{s}_1 + \widehat{s}_2$ and $\widehat{b} = \widehat{b}_1 + \widehat{b}_2$. The analysis of unbiasedness carries through because we can consider that the bin is actually two sub-bins, one ending when the first record ends and the other starting at the same time. Since we have unbiased byte and packet estimates for both sub-bins, our estimates for the sum of the bins will still be unbiased.

If $c_{s1} > 0$, we know that the flow sent packets during the bin, so we set $\widehat{f}$ to 1, otherwise we use Equation 3 with c_{s2} since an unbiased estimator for whether the flow was active in the second sub-bin will tell use whether it was active overall. This approach preserves overall unbiasedness, but it makes analysis more complicated because the two flow records representing the flow cannot be processed independently anymore: the contribution of the second record to the flow count of the bin depends on whether there was a first record with the same flow identifier. When the router reports the records, they might not be near each other, so the analysis has to do "flow reconstruction": keep a hash table with flow identifiers and find flow records with the same flow identifier covering parts of the same bin. The consequence of not doing flow reconstruction is running the risk of double counting such flows with more than one record (which might be acceptable in many settings).

By our definition of flow arrivals from Section 4.4, as long as assumption 1 holds, if a flow has a record that starts before the start of the bin, we should use $\widehat{A} = 0$, irrespective of whether we have a second flow record (possibly with a SYN flag) or not. If we have a second flow record with the SYN flag set we can clearly say that assumption 1 does not hold, but if we do not do flow reconstruction we might count it separately against the flow arrival count. In many setting this type of overcounting is not a big concern.

4.7.2 Slices shorter than bins

When the inactive timeout $t_{inactive}$ is short or when the analysis is over long time bins (say hours), flow slices can be shorter than the bin size. It can happen that we have more than two records for the same flow within the same bin. For byte and packet counts we can just add the individual estimates for the different records and we get an unbiased estimator for the entire bin. For active flows we cannot get an unbiased estimate, not even with flow reconstruction. For flow arrivals, by using $\widehat{A}^{(1)}$ for the individual records[6] and summing the contributions without any flow reconstruction gives unbiased estimates as long as assumption 1 is not violated.

4.7.3 Binning errors

So far we assumed that Flow Slices uses binned measurement. This guarantees that as long as the analysis is on time intervals that are exact multiples of the measurement bins used, it will be easy to determine exactly how many of the packets and the bytes counted by the record were within the bin. But by default Flow Slices doesn't use bins, and for records that span bin boundaries, the user will have to guess how the packets and bytes were actually divided between the bins. We can prove that our reconstruction of how the traffic divides between the bins is unbiased only if we make an assumption about the spacing of the packets.

Assumption 3 For every flow at the input of the flow slicing algorithm, the time between the arrivals of all

[6] For a record started before the beginning of the bin, even if it has the SYN flag set we consider that the SYN packet was one of the flow's packets that arrived before the beginning of the bin and thus have $\widehat{A}^{(1)} = 0$.

pairs of its consecutive packets is the same.

We use the following algorithm for distributing the packets of reported by a flow record that spans bins between the bins covered by the record. We consider c_s packet arrival events, the first one is the timestamp of the first packet counted by the entry, the last one the timestamp of the last packet counted by the entry and the remaining $c_s - 2$ evenly spaced between them. We consider that 1 packet arrived at every packet arrival event, except for the first event which has $1/p$ packets, and distribute the packets between bins accordingly. This can be shown to be an unbiased way of distributing packets between bins under assumption 3. We recommend distributing the c_b bytes of the flow between bins proportionally with the number of packets counted against each bin. Assumption 3 is not enough to prove this distribution of bytes between the bins to be unbiased, we would need an additional assumption about uniformity of packet sizes. For flow arrivals, we do not have a binning problem because we assume that the first packet counted by the flow record is the one with the SYN, so we count the flow arrival against the bin the first packet is in.

We cannot achieve provably unbiased binning for bytes and packets under realistic assumptions about inter packet arrival times and packet size distributions within flows. We turn to measurements instead to see how much the binning error is on typical traffic. We recommend using such experimental results to decide whether increasing the size of the flow record by adding multiple counters to do binned measurement is worth it.

5 Variances of estimators

The estimators discussed in the previous section were all defined on an individual flow and to compute a measure (say the number of packets) for a larger aggregate, the analyst would sum the values of the estimators for the flow records matching the aggregate. The sampling decisions for different flows are fortunately independent[7] and thus the variance of the estimates for aggregates are the sum of the respective variances for the estimators for individual flows. In this section we focus on studying the variances of the various estimators for individual flows. We also show that the variances of the estimators based on the core flow slicing algorithm are lower than those of estimators based on random sampling used by Adaptive NetFlow to control memory usage. As in Section 4, we start with a simplified setting of a single bin in isolation and then proceed to more realistic settings. The proofs for the variance results from this section can be found aprefvarianceproofs.

5.1 Packet count variance

For the core flow slicing algorithm we can compute the variance of the packet count estimator.

$$VAR[\hat{s}] = 1/p(1/p - 1)(1 - (1 - p)^s) \tag{7}$$

Note how this variance is strictly lower than the variance of results based on random packet sampling $(1/p - 1)s$ except for the case of $s = 1$ when the two variances are equal since in this case both algorithms have probability p of estimating the packet count as $1/p$ and probability $1 - p$ of estimating it as 0. The higher s, the larger the difference between the variance of results based on flow slicing when compared with packet sampling. Since using the same sampling probability will give the same memory usage for flow slicing and ordinary sampling, this comparison of variances shows us that flow slicing is a superior solution. The advantage is most apparent when estimating the traffic of aggregates with much traffic coming from large flows.

The same conclusion holds if we compare the combination of packet sampling and flow slicing used by Flow Slices to the pure packet sampling used by Adaptive NetFlow and Sampled NetFlow. Here the fair comparison is with Sampled NetFlow using a packet sampling probability of pq. We can conceptually divide this into a first stage of packet sampling that samples packets with probability q and a second one that samples them with probability p. The first stage has identical statistical properties for the two solutions, thus the difference in the accuracy is given by the second stage, but comparing the second stages reduces to comparing flow slicing and packet sampling using the same probability p.

5.2 Byte count variance

We can also compute the variance of the estimates for the number of bytes[8].

$$VAR[\hat{b}] = 1/p \sum_{i=1}^{s} (1 - p)^{i-s+1} b_i^2 \tag{8}$$

Note how this variance is strictly lower than the variance of results based on random packet sampling $(1/p - 1)\sum_{i=1}^{s} b_i^2$ (except for the case of a single packet flow). This shows that for byte counts too, flow slices are a better solution than ordinary sampling.

5.3 Flow count variance

We can also compute the variance of the estimates for the number of active flows, but we cannot compare against packet sampling because there are no unbiased estimates for the number of active flows based on packet sampled data.

[7] Strictly speaking once we add algorithms that adapt various sampling parameters dynamically based on resource consumption we introduce small correlations between decisions, but these correlations are so small we can safely ignore them.

[8] Remember that we number the packet sizes b_i in reverse order with b_1 being the size of the last packet and b_s that of the first one.

$$VAR[\widehat{f}] = (1-p)^{s-1}(1/p-1) \qquad (9)$$

5.4 Continuous operation

If we consider continuous operation for the algorithm, we can have at the beginning of the bin a record for our flow. If the slice spans the entire bin, it counts everything exactly and thus the variance of all estimator is 0. If the slice ends in the current bin, we can divide the flow into two parts: one covered by this older record and the rest. For the first part we have 0 variance for the byte and packet counts and for the second part we can apply formulas 7 and 8, but instead of s being the number of packets of the flow in the bin, it should be only the number of packets in this second part and the b_i be the sizes of those packets. For the flow count estimate, if the number of packets in the first record is 0 (whether it is 0 or not is not something that depends on the random flow slicing decisions in the current bin), the variance of the estimate is 0, otherwise formula 9 applies. Thus having flow records active at the beginning of the bin does not increase the variance of the packet, byte and flow count estimates, on the contrary, it can reduce them significantly.

6 Experimental evaluation

We divide the experimental evaluation section into two parts. In the first group of experiments, we evaluate the efficacy of the core flow slicing algorithm. Later, we compare flow slicing with Adaptive NetFlow to show the efficacy of Flow Slices both in terms of memory usage and accuracy of estimates. For our evaluations, we obtained real OC-48 traces from Cooperative Association of Internet Data Analysis (CAIDA[4]).

6.1 Accuracy of the core flow slicing algorithm

In this section, we evaluate the core flow slicing algorithm against the "full-state" approach. These experiments provide more insight into the efficacy of the flow slicing algorithm and the reaction to changing various variables such as flow slicing probability, slice length on the memory usage and the mean relative error.

First, we fixed the slicing probability to 0.008 (equal to 1 in 125 flows) and the slice duration to 60 seconds. Figure 2 shows the scatter plot of ratio of the estimated flow size (in number of packets) and the actual true flow slice on the y-axis and the true flow size on the y-axis. Note that the plot only shows flows that have more than 5000 packets throughout the duration of the trace (1 hour). From this scatter plot, we can see that most of the flows have been accurately estimated within 10% error margin. Also, as the flow sizes became bigger, the estimate converges to the true estimate as these flows are more rapidly sampled and once sampled counted fully. Note also the presence of two-sided errors clearly depicting the "unbiased-ness" of our estimates using Flow Slices.

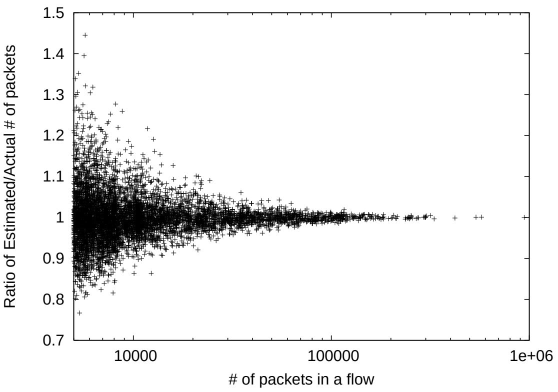

Figure 2: Scatter plot for accuracy of flow slices.

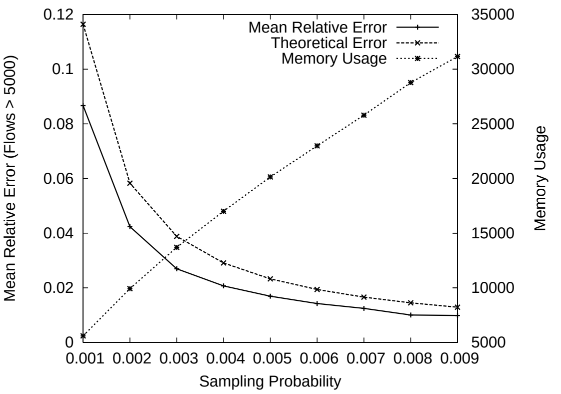

Figure 3: Trade-off between Mean Relative Error and Memory Usage as we increase sampling probability

What is the affect of flow slicing probability on the accuracy of these estimates ? According to the theory in Section 5.1, increasing slicing probability increases the accuracy of estimated flow sizes. In other words, the mean relative error as defined as ratio of the mean of the error to the actual value should decrease. Also, clearly as the slicing probability increases, the memory usage should increase almost linearly. In Figure 3, the mean relative error for flows larger than 5000, and the corresponding memory usage have been plotted with varying slicing probability on the x-axis. Apart from the empirical value of the mean relative error, we also plot the theoretical value for this based on the formula obtained in Section 5.1. From this figure, we can see that the results are as calculated theoretically. Increasing slicing probability decreases the mean relative error although amount of memory usage increases almost linearly.

Extrapolating Bins from Flow Slices: The goal of this experiment is to study the affect of binning from flow

Flow Size	ANF		Flow Slices(60s)		Flow Slices (180s)		Flow Slices (300s)	
	Pkts.	Bytes	Pkts.	Bytes	Pkts.	Bytes	Pkts.	Bytes
> 1%	0.025	0.048	0.023	0.021	0.02	0.020	0.0140	0.038
0.1-1%	0.113	0.158	0.06	0.079	0.055	0.064	0.045	0.059
0.01-0.1%	0.31	0.406	0.21	0.303	0.183	0.265	0.179	0.244
Web (80)	0.0121	0.0464	0.0074	0.0177	0.0215	0.0101	0.0071	0.0567
Mail (25)	0.0003	0.0326	0.0670	0.0376	0.0141	0.0307	0.0176	0.0252
SSH (22)	0.1894	0.1916	0.1033	0.5267	0.0020	0.0381	0.0088	0.5670
News (119)	0.0381	0.0167	0.0214	0.0139	0.0032	0.0149	0.0001	0.0028
FTP (20)	0.0294	0.0233	0.0475	0.0005	0.0238	0.0123	0.0485	0.1061

Table 4: Results comparing Adaptive Netflow and Flow Slices with different Slice durations. The total number of packets are about 35 Million.

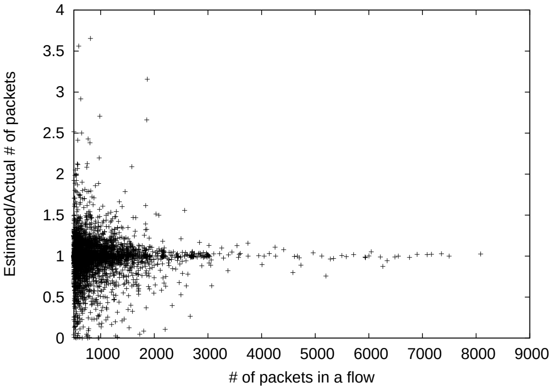

Figure 4: Scatter Plot that depicts the errors introduced in extrapolating bin measures from slices.

slices. We plot in Figure 4, the ratio of estimated to the actual size of the flow in a given bin to the flow size on x-axis. For this experiment, we used a slice length of 90 seconds and divided it up equally into 10 bins of size 9 seconds each. Figure 4 shows that larger the flows, binning error is insignificant. However, for relatively smaller flows extrapolating from flow slices results in much higher error. Since we divide up the entire volume of traffic for a particular flow equally among all the bins (except the first bin which gets a slightly higher amount), error can be dependent on the burstiness of the traffic. Of course, to capture the fine grained traffic slices, the extension proposed in Section 3.4 could be applied but that would result in higher memory requirement. Another important thing to note here is the unbiased-ness of the estimate as we can see two-sided errors.

6.2 Comparison with Adaptive NetFlow

In this subsection, we compare Flow Slices with Adaptive NetFlow, a previously proposed solution that is based on packet sampling. For the purposes of evaluation, we fix the packet sampling probability to 1 in 1024 for Adaptive NetFlow. To be fair in our comparisons with Flow Slices, we split the $1/1024$ probability into two parts consisting of packet sampling ($1/16$ for our OC-48) and flow slicing probability ($1/64$). We picked a random 5 minute OC-48 trace obtained from CAIDA for our comparisons. Also, instead of monitoring individual flows, we aggregated based on the destination IP address in the flow as they tend to be much larger and hence significantly farther from statistical noise. This also allows for a fairer comparison between the two schemes as the final aggregates instead of individual flows are usually most important for traffic analysis.

Table 4 illustrates the comparison of error obtained by Adaptive NetFlow and Flow Slices both for packet counts and byte counts. Clearly, in the first group of flows that are larger than 1% of the total traffic volume, Flow Slices performs slightly better than Adaptive NetFlow. When we used the slice length of 300s, we found that Flow Slices has about 11% less mean relative error than that of Adaptive NetFlow. We believe this is due to the fact that once a flow is sampled by Flow Slices, it remains in the memory until the slice expires hence leading to more accurate results. In the second group of flows that contained traffic volume between 0.1% and 1% of total traffic, once again Flow Slices provide better accuracy than Adaptive NetFlow by about 4-7%. Finally, as expected for really small flows, sample and hold based algorithms perform better than ordinary sampling and we can see that Flow Slices performs better than Adaptive NetFlow by almost 7-13%.

In the second part of the Table 4, we show how Adaptive NetFlow and Flow Slices estimated the individual traffic breakdown for common traffic types such as WWW, Email etc. Both Flow Slices and Adaptive NetFlow estimated close to the actual packet counts for Mail, News, Web traffic, SSH and FTP. For SSH, the case when slice duration was 60 seconds and 300 seconds had significant error but, slice length of 180 seconds produced more accurate byte counts. Too few connections (only

Slice	Memory	Volume	ANF
60	7122	21056	21526
180	15917	22979	21526
300	21587	21587	21526

Table 5: Comparing memory used and volume of records generated by Flow Slices and Adaptive NetFlow (ANF). Here, we used 300 seconds for bin size of Adaptive NetFlow. We did not use any inactivity time-out for flow slices here.

Slice/	Memory		Volume	
Bin	Slices	ANF	Slices	ANF
60	3233	5484	27589	27491
180	4022	13983	24602	23859
300	4617	21526	23398	21526

Table 6: Memory used and Volume of Records generated by Flow Slices and Adaptive NetFlow (ANF) for similar Adaptive NetFlow binning and Flow Slice durations. We used an inactivity timeout of 15 seconds for these experiments

74) found in the trace, coupled with very little volume (0.03% traffic) could be attributed to this error in accuracy. In general, however, we can see that byte counts and packet count errors are fairly low to show that flow slices helps obtain accurate estimates to flows. Unbiased errors statistically equate out as the constituent number of flows increases as well as size of the aggregate.

Memory Requirements: The total volume of flow records generated by Adaptive NetFlow and Flow Slices was found to be roughly comparable. Adaptive NetFlow generated about 21526 records, while Flow Slices depending on the slice length, generated about 21000 to 24000 records. However, the key gain that Flow Slices have in comparison to Adaptive NetFlow is in the area of run time memory. We saw that if we used 60 seconds as the slice length, Flow Slices operate within a third of the number of records that Adaptive NetFlow requires thus making it more memory efficient than Adaptive NetFlow. The second key observation from Table 5 is the fact that the total volume of records output by Adaptive NetFlow and Flow Slices is roughly comparable. This is expected since both Adaptive NetFlow and Flow Slices are run with similar final probabilities (1 in 1024).

Effect of Inactivity Timeouts: In Table 6, we show the effect of introducing "Inactivity Timeouts" on the memory usage for the Flow Slices algorithm. A flow record that sees no activity for a pre-defined inactivity period is immediately flushed out of the memory. The inactivity timeout we used for this experiment is 15 seconds. So, short flows that last for less than 15 seconds typically get flushed out much faster than the rest of the flows thus saving memory usage. As expected, with this inactivity timeout, we see that Flow Slices gain an order of magnitude memory savings in comparison with Adaptive NetFlow. Note also the slight increase in run-time memory as we increase the slice length relative to the case when inactivity timeouts are not used (Compare column 2 of Table 5 and Table 6). The reason is that only really long flows tend to occupy space when we increase the slice length. Short flows are not affected by the slice duration. Since the number of long flows tends to be small, memory is re-used more efficiently than when inactivity timeouts are not applied. For Adaptive NetFlow, reducing the bin size has a similar affect that increases the volume of flow records but reduces the operational memory footprint. Clearly, for comparable volumes of flow records, Flow Slices operate with a much smaller memory footprint when inactivity timers are enabled. This is much more pronounced when we use larger values of slice lengths. For example, when a slice length of 300 is used, we see that Flow Slices generate only 10% more flow records, but operates with a memory footprint 5 times smaller than the Adaptive NetFlow counterpart.

From these results, we have empirically verified the efficacy of the Flow Slices in comparison with Packet Sampling based algorithms such as Adaptive NetFlow. When we apply inactivity timeouts to the Flow Slices, it results in much better spatial re-use of memory while suffering little loss in accuracy and little increase in the total volume of flow records.

7 Conclusions and future work

Processing, memory, and bandwidth constraints make it impossible for high speed routers to provide full flow measurements thus forcing us to consider some type of data reduction. Different flow measurement solutions perform this data reduction differently, and one can compare them by comparing their resource consumption and the amount of error the data reduction causes in various analyses one wants to perform on the flow data. We motivated our design of Flow Slices with the desire to support accurate estimates for the number bytes, packets and flows in arbitrary large aggregates within the traffic.

Flow Slices offer a unique mix of qualities among flow measurement solutions: dynamic adaptation of sampling parameters to keep resource usage within limits, separate parameters for controlling the three potential resource bottlenecks, efficient use of available resources, and algorithmic solutions for minimizing the errors introduced by the data reduction. These qualities are possible due to novel algorithms such as the core flow slicing algorithm and multifactor smart sampling and various new estimators. Our experiments also confirm that compared to the currently used Sampled NetFlow and to another solution that can be deployed by a simple software upgrade at routers, Adaptive NetFlow, Flow Slices constitute a better flow measurement solution.

But the fact that Flow Slices support well the traffic analyses discussed in this paper, does not mean there is no room for improvement. There are many useful analyses of unsampled flow data that we haven't considered. For example correlation between various flows has been used to classify data transfers: the existence of a control connection on the ftp port between two IP addresses can help identify a highport to highport connection as a passive ftp transfer, the existence of a prior connection to the central Napster servers has been used to identify subsequent highport to highport connections as Napster traffic [19], connections to computers that use well known peer to peer ports and the existence of both UDP and TCP connections between computers have been used to identify highport to highport p2p traffic [13]. Additional metrics such as flow duration and the variability of packet inter arrival times have been used to divide flows into different application categories [21]. We are confident that progress in data reduction solutions by traffic measurement solutions for high speed links might someday enable these and many other useful analyses and turn the Internet into a better understood and more reliable network.

8 Acknowledgments

We wish to thank Nick Duffield for discussions that contributed to the idea of the $\widehat{f}$ estimator and Ken Keys for helping us with the code of Adaptive NetFlow.

References

[1] Ipmon - packet trace analysis. http://ipmon.sprintlabs.com/ packstat/ packetoverview.php.

[2] N. Brownlee, C. Mills, and G. Ruth. Traffic flow measurement: Architecture. RFC 2722, Oct. 1999.

[3] N. Brownlee and D. Plonka. IP flow information export (ipfix). IETF working group.

[4] Cooperative association for internet data analysis. http://www.caida.org/.

[5] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction: How much is enough? In *Proceedings of the ACM SIGMOD*, 1998.

[6] C. Cranor, T. Johnson, O. Spatschek, and V. Shkapenyuk. Gigascope: A stream database for network applications. In *p-sigmod*, June 2003.

[7] The DAG project. http://dag.cs.waikato.ac.nz/.

[8] N. Duffield, C. Lund, and M. Thorup. Charging from sampled network usage. In *SIGCOMM Internet Measurement Workshop*, Nov. 2001.

[9] N. Duffield, C. Lund, and M. Thorup. Properties and prediction of flow statistics from sampled packet streams. In *SIGCOMM Internet Measurement Workshop*, Nov. 2002.

[10] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better netflow. In *Proceedings of the ACM SIGCOMM*, Aug. 2004.

[11] C. Estan and G. Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. In *ACM Trans. Comput. Syst.*, Aug. 2003.

[12] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True. Deriving traffic demands for operational ip networks: Methodology and experience. In *Proceedings of the ACM SIGCOMM*, pages 257–270, Aug. 2000.

[13] T. Karagiannis, A. Broido, M. Faloutsos, and K. claffy. Transport layer identification of p2p traffic. In *Internet Measurement Conference*, Oct. 2004.

[14] K. Keys, D. Moore, R. Koga, E. Lagache, M. Tesch, and k claffy. The architecture of CoralReef: an Internet traffic monitoring software suite. In *PAM2001 — A workshop on Passive and Active Measurements*. CAIDA, RIPE NCC, Apr. 2001. http://www.caida.org/outreach/papers/2001/CoralArch/.

[15] R. R. Kompella and C. Estan. The power of slicing in internet flow measurement. Technical report, UCSD Technical Report, May 2005.

[16] K. McCloghrie and M. T. Rose. Rfc 1213, Mar. 1991.

[17] Cisco NetFlow. http://www.cisco.com /warp /public /732 /Tech /netflow.

[18] V. Paxson. Bro: a system for detecting network intruders in real-time. In *Computer Networks (Amsterdam, Netherlands: 1999)*, volume 31, pages 2435–2463, 1999.

[19] D. Plonka. Flowscan: A network traffic flow reporting and visualization tool. In *USENIX LISA*, pages 305–317, Dec. 2000.

[20] M. Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th Systems Administration Conference*. USENIX, 1999.

[21] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield. Class-of-service mapping for QoS: A statistical signature-based approach to IP traffic classification. In *Internet Measurement Conference*, Oct. 2004.

[22] Sampled NetFlow. http://www.cisco.com/ univercd/cc/td/ doc/product/software/ ios120/120newft/ 120limit/120s/120s11/12s_sanf.htm.

A Proofs of variance results

A.1 Proof of Equation 7

$$VAR[\widehat{s}] = 1/p(1/p-1)(1-(1-p)^s)$$

We first prove that $E[\widehat{s}^2] = 1/p(1/p-1)(1-(1-p)^s) + s^2$ by induction on s.

Base case If s=1, $E[\widehat{s}^2] = p(1/p)^2 + 0 = 1/p$. Also $1/p(1/p-1)(1-(1-p)^1) + 1^2 = 1/p - 1 + 1 = 1/p$.

Inductive step By induction hypothesis we know that for $s' = s-1$, $E[\widehat{s'}^2] = 1/p(1/p-1)(1-(1-p)^{s'}) + s'^2 = 1/p(1/p-1)(1-(1-p)^{s-1}) + (s-1)^2$.

$$\begin{array}{rcl}
E[\widehat{s}^2] & = & p(1/p-1+s)^2 + (1-p)E[\widehat{s}'^2] \\
 & = & 1/p + 2(s-1) + p(s-1^2) + (1-p)(s-1)^2 \\
 & & +(1-p)1/p(1/p-1)(1-(1-p)^{s-1}) \\
 & = & 1/p - 1 + 1 + 2(s-1) + (s-1^2) \\
 & & +(1-p)1/p(1/p-1) - 1/p(1/p-1)(1-p)^s \\
 & = & 1/p(1/p-1)(p+1-p) + s^2 \\
 & & -1/p(1/p-1)(1-p)^s \\
 & = & 1/p(1/p-1)(1-(1-p)^s) + s^2
\end{array}$$

$$VAR[\widehat{s}] = E[\widehat{s}^2] - E[\widehat{s}]^2 = 1/p(1/p-1)(1-(1-p)^s)$$

A.2 Proof of Equation 8

$$VAR[\widehat{b}] = 1/p\sum_{i=1}^{s}(1-p)^{i-s+1}b_i^2$$

We prove this by induction on the number of packets s.

Base case For $s=1$, the packet is sampled with probability p, in which case the estimate of the number of bytes is $\widehat{b} = b_1/p$, otherwise $\widehat{b} = 0$. $VAR[\widehat{b}] = E[\widehat{b}^2] - E[\widehat{b}]^2 = pb_1^2/p^2 - b_1^2 = (1-p)/pb_1^2$.

Inductive step From the induction hypothesis we know that if the first packet is not sampled, the variance in the estimate $\widehat{b'}$ of the number of bytes in the next $s-1$ packets is $VAR[\widehat{b'}] = E[\widehat{b'}^2] - E[\widehat{b'}]^2 = 1/p\sum_{i=1}^{s-1}(1-p)^{i-s}b_i^2$.

$$\begin{array}{rcl}
VAR[\widehat{b}] & = & E[\widehat{b}^2] - E[\widehat{b}]^2 \\
 & = & p(b_s/p + b')^2 + (1-p)E[\widehat{b}'^2] - (b_s + b')^2 \\
 & = & pb_s^2/p^2 + 2b_sb' + pb'^2 - b_s^2 - 2b_sb' - b'^2 \\
 & & +(1-p)E[\widehat{b}'^2] \\
 & = & (1/p-1)b_s^2 + (1-p)(E[\widehat{b}'^2] - b'^2) \\
 & = & (1-p)/pb_s^2 + (1-p)(E[\widehat{b}'^2] - E[\widehat{b}']^2) \\
 & = & (1-p)/pb_s^2 + 1/p\sum_{i=1}^{s-1}(1-p)^{i-s+1}b_i^2 \\
 & = & 1/p\sum_{i=1}^{s}(1-p)^{i-s+1}b_i^2
\end{array}$$

A.3 Proof of Equation 9

Let $p_{s-1} = 1-(1-p)^{s-1}$ be the probability that flow slicing selects one of the packets of the flow before the last one.

$$\begin{array}{rcl}
E[\widehat{f}^2] & = & p_{s-1}\cdot 1 + (1-p_{s-1})(p(1/p)^2 + (1-p)\cdot 0) \\
 & = & p_{s-1} + (1-p_{s-1})1/p \\
VAR[\widehat{f}] & = & E[\widehat{f}^2] - E[\widehat{f}]^2 = p_{s-1} + (1-p_{s-1})1/p - 1 \\
 & = & (1-p_{s-1})(1/p-1) = (1-p)^{s-1}(1/p-1)
\end{array}$$