Building a Better NetFlow

Cristian Estan[*] cestan@cs.ucsd.edu
Ken Keys[†] kkeys@caida.org
David Moore[*,†] dmoore@caida.org
George Varghese[*] varghese@cs.ucsd.edu

ABSTRACT

Network operators need to determine the composition of the traffic mix on links when looking for dominant applications, users, or estimating traffic matrices. Cisco's NetFlow has evolved into a solution that satisfies this need by reporting flow records that summarize a sample of the traffic traversing the link. But sampled NetFlow has shortcomings that hinder the collection and analysis of traffic data. First, during flooding attacks router memory and network bandwidth consumed by flow records can increase beyond what is available; second, selecting the right static sampling rate is difficult because no single rate gives the right tradeoff of memory use versus accuracy for all traffic mixes; third, the heuristics routers use to decide when a flow is reported are a poor match to most applications that work with time bins; finally, it is impossible to estimate without bias the number of active flows for aggregates with non-TCP traffic.

In this paper we propose Adaptive NetFlow, deployable through an update to router software, which addresses many shortcomings of NetFlow by dynamically adapting the sampling rate to achieve robustness without sacrificing accuracy. To enable counting of non-TCP flows, we propose an optional Flow Counting Extension that requires augmenting existing hardware at routers. Both our proposed solutions readily provide descriptions of the traffic of progressively smaller sizes. Transmitting these at progressively higher levels of reliability allows graceful degradation of the accuracy of traffic reports in response to network congestion on the reporting path.

Categories and Subject Descriptors: C.2.3 [Computer-Communication Networks]: Network Operations – *Network monitoring*

General Terms: Algorithms, Measurement.

Keywords: Traffic measurement, Network monitoring, Data summarization.

[*]CSE Dept., University of California, San Diego
[†]CAIDA, University of California, San Diego

1. INTRODUCTION

Traffic measurement is crucial to operating all IP networks because networks must be provisioned based on the traffic they carry. Flow level measurements are also widely used for security reasons or to provide insight into the traffic crossing a network. Many existing systems that examine finer details of traffic to reveal malicious activities [25, 28], monitor complex performance metrics [10] or capture unsampled traces of traffic. However, these systems based on unsampled traces have inherent scalability problems that restrict their deployment to lower speed links. SNMP counters [21] are a simpler solution more widely deployed than flow level measurement, but they fail to give details about the composition of the traffic mix as they report only the total amount of traffic transmitted on the measured link.

Sampled flow level measurement provides a balance between scalability and detail because performance limits can be addressed by reducing the sampling rate. Thus it is no surprise that Cisco's NetFlow [24] (and other compatible flow measurement solutions implemented by major router manufacturers, some under standardization by IETF [5, 6, 4]) are widely deployed and constitute the most popular way of measuring the composition of network traffic. Most major ISPs rely on NetFlow data to provide input to traffic analysis tools that are widely used by network operators. NetFlow data is also used in computer networking research.

While the wide deployment and use of NetFlow is proof of its ability to satisfy important needs of network operators, it is not an indication that it cannot be improved. In this paper we identify several shortcomings of NetFlow, and propose evolutionary solutions to these problems that are backwards compatible and support incremental deployment.

The main contributions of this paper are as follows.

- **1. Sampling Rate Adaptation:** NetFlow uses a *static sampling* rate which is either suboptimal at low traffic volumes or can cause resource consumption (memory, bandwidth) difficulties at high traffic volumes. Our adaptive algorithm, by contrast, provably stays within fixed resource consumption limits while using the optimal sampling rate for all traffic mixes.

- **2. Renormalization of flow entries:** We introduce a new idea in traffic measurement called *renormalization* which allows us to reduce the number of NetFlow entries after a decrease in sampling rate. We introduce efficient algorithms for renormalization that make adapting the sampling rate feasible. Renormalization also enables a layered transmission of NetFlow data that gracefully degrades the accuracy of traffic reports in response to network congestion on the reporting path.

- **3. Time bins:** Most traffic analysis tools divide the traffic stream into fixed intervals of time that we call *bins*. Unfortunately, NetFlow records can span bins, causing unnecessary complexity and inaccuracy for traffic analysis. Our Adaptive NetFlow, by contrast, ensures that flow records do not span bins. This simple idea is *essential* in order to provide statistical guarantees of accuracy after operations such as renormalization and sampling rate adaptation.

- **4. Accurate flow counting:** It is well known that Sampled NetFlow cannot give accurate counts of non-TCP flows. Such counts are important for detecting attacks (e.g., Slammer worm) and scans. While our previous contributions require only software changes, we show how a modest and easily implementable hardware addition (which we call the Flow Counting Extension) can give accurate flow counts even for non-TCP flows. Our extension configured to report just 8000 entries provides better results *even for TCP flow counts* than SYN counting estimators based on NetFlow reports of 64K entries.

The organization of this paper is as follows. We provide an introduction to NetFlow in Section 1.1 and describe some major problems of NetFlow in Section 1.2. We survey related work in Section 1.3. Next, in Section 2 we present our Adaptive NetFlow (ANF) proposal, which solves many of NetFlow's problems (using adaptation, renormalization and time bins) and is deployable through a simple update to the router software. In Section 3 we propose an optional Flow Counting Extension (FCE) which solves the problem of getting accurate flow counts for non-TCP flows, but changes to the router hardware are required. Finally, in Section 4 we present our experimental evaluation of ANF and FCE.

1.1 NetFlow

NetFlow [24], first implemented in Cisco routers, is the most widely used flow measurement solution today. It started as a cache for improving the performance of IP lookups and was later adapted to flow measurement. Routers running NetFlow maintain a "flow cache" containing flow records that describe the traffic forwarded by the router. These flow records are then exported using unreliable UDP to a computer that collects, analyzes and archives them.

For each router interface, flows are identified by important fields in the packet header: source and destination IP address, protocol, source and destination port, and type of service byte. The router inserts a new flow record into the flow cache if a packet does not belong to an existing flow. NetFlow uses four rules to decide when a flow has ended which then allows the corresponding record to be exported: 1) when indicated by TCP flags (FIN or RST), 2) 15 seconds (configurable) after seeing the last packet with a matching flow ID, 3) 30 minutes (configurable) after the record was created (to avoid staleness) and 4) when the flow cache is full. Besides the fields identifying the flow, each flow record also keeps other data such as the number of packets and bytes in the flow and the timestamps of the first and last packet. These records allow many kinds of analyses. For example, using the port numbers present in the exported flow

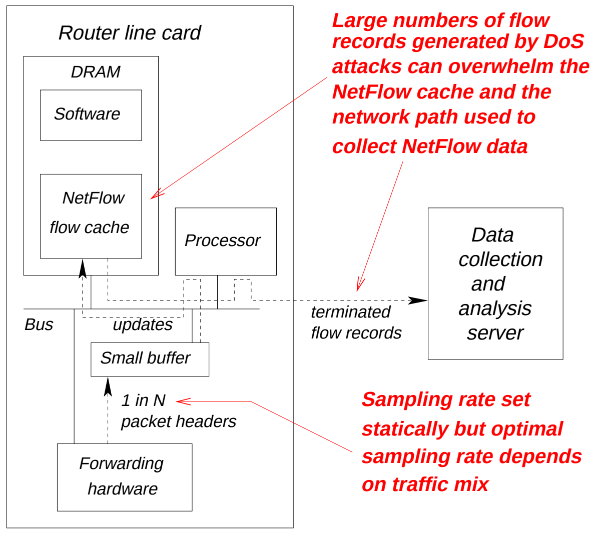
Router line card
DRAM
Software
NetFlow flow cache
Processor
Bus
updates
Small buffer
1 in N packet headers
Forwarding hardware
terminated flow records
Data collection and analysis server
Large numbers of flow records generated by DoS attacks can overwhelm the NetFlow cache and the network path used to collect NetFlow data
Sampling rate set statically but optimal sampling rate depends on traffic mix

Figure 1: *Problems: number of records strongly depends on traffic mix and network operator must set sampling rate.* With unfriendly traffic mixes, the number of flow records generated by NetFlow increases significantly and this can exhaust the memory at the router and the bandwidth available for reporting the records to the collection station. Setting NetFlow's sampling rate is hard because the optimal sampling rate depends on the traffic mix.

records, an analyst can produce a breakdown of the traffic by application; using the IP addresses, one can produce a traffic breakdown by source or destination [27]. By combining data from multiple routers one can obtain a network-wide view of the traffic demands of the ISP's customers [16].

To update the NetFlow cache when a packet is seen, NetFlow must look up the corresponding entry in the flow cache (creating a new entry if necessary) and update that entry's counters and timestamps. Since for high speed interfaces, the processor and the memory holding the flow cache cannot keep up with the packet rate, Cisco introduced sampled NetFlow [29] which updates the flow cache only for sampled packets. For a configurable value of a parameter N, one of every N packets is sampled. When using sampled NetFlow records analysts compensate for the sampling by multiplying recorded values by N, the inverse of the sampling rate.

1.2 Problems with NetFlow

Even though it is widely used, NetFlow has problems. In this paper we identify and address four of them.

- **Number of records strongly depends on traffic mix.** A larger than expected number of records can overwhelm the router and the network path to the collection station, as illustrated by Figure 1. Today's traffic mixes often include massive flooding denial of service attacks or aggressive port and IP scans that generate a large number of "flows" consisting of a single small packet. The number of entries exported under these circumstances is very large, and the traffic they generate can cause the network to drop packets. Duffield and Lund show [11] that the errors in-

Problem	Solution	Requirement
Number of records strongly depends on traffic mix (Figure 1)	Adapting sampling rate to traffic (Section 2.2)	software update
Network operator must set sampling rate (Figure 1)		
Mismatch between flow termination heuristics and analysis (Figure 2)	Measurement bins (Section 2.1)	software update
Cannot estimate the number of flows (Figure 3)	Sampling flows (Section 3)	hardware addition

Table 1: Summary of NetFlow problems and proposed solutions.

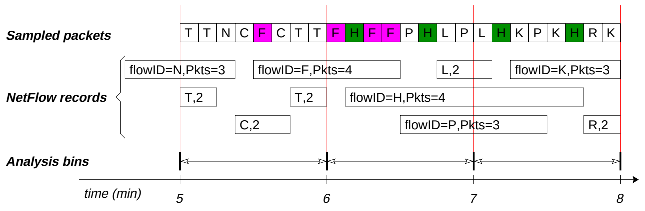

Figure 2: *Problem: mismatch between flow termination heuristics and analysis.* The heuristics used by NetFlow to terminate flow records do not match the time bin model used by traffic analysis. For flow records that span multiple bins, the analysis application has to estimate how many of the packets reported belong to each bin. While assuming the packets were uniformly distributed can give good results as for flow record H it often produces inaccurate ones as for flow F.

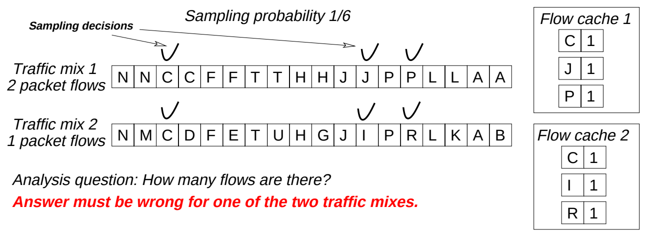
Sampling decisions
Sampling probability 1/6
Traffic mix 1
2 packet flows
N N C C F F T T H H J J P P L L A A
Traffic mix 2
1 packet flows
N M C D F E T U H G J I P R L K A B
Analysis question: How many flows are there?
Answer must be wrong for one of the two traffic mixes.
Flow cache 1
C 1
J 1
P 1
Flow cache 2
C 1
I 1
R 1

Figure 3: *Problem: cannot estimate the number of flows* Assume we want to estimate the number of flows in these two traffic mixes with the same number of packets and identical sampling decisions. Note that the first mix contains 2 packet flows and the second one twice as many 1 packet flows. Both flow caches contain three flow records and each has a packet count of 1, so whatever estimator we use we will get the same answer for both cases. It will be significantly off for at least one of the traffic mixes.

troduced by lost NetFlow packets are worse than the errors introduced by various types of intentional sampling within the measurement infrastructure. To solve this problem we propose in Section 2.2 adapting the sampling rate to the traffic mix.

- **Network operator must set sampling rate.** Setting the sampling rate involves tradeoffs. The lower the sampling rate, the fewer the packets that are sampled. This reduces the load of the processor running NetFlow and the strain on router memory and on the network used to export the flow records. But a lower sampling rate also means larger errors in traffic measurement and analysis. The sampling rate constituting the best compromise between these two opposing considerations depends on the traffic mix: when the traffic is low we want a higher sampling rate to obtain better accuracy, while when the volume of the traffic is high and when massive attacks are in progress we need a lower sampling rate to protect the measurement infrastructure. Setting the static sampling rate is a hard decision for the network operator. To spare operators this dilemma we propose adapting the sampling rate to the traffic mix in Section 2.2.

- **Mismatch between flow termination heuristics and analysis.** Traffic analysis and visualization [16, 27, 3, 1, 19] groups traffic into time intervals usually referred to as "bins". The size of these bins ranges from minutes to days and most often one analyzes many consecutive bins of the same size. If the timestamps indicating the start and the end of the NetFlow record are within a single bin, all the packets of the flow are counted against that bin and processing is simple. On the other hand if the flow starts in one bin and finishes in another, one needs to estimate how much of the traffic belonging to the flow went to each bin. This complicates processing and introduces inaccuracies, as shown in Figure 2. Furthermore, interactions between sampling and the flow termination heuristics can lead to flow splitting such as for flow T in Figure 2 which increases the number of times the flow is reported [11]: because NetFlow only sees the sampled packets, the time between consecutive packets can increase to more than the "inactive timer" (15 seconds by default) and NetFlow terminates the record prematurely and reports fragments of the flow separately. To solve this problem we propose in Section 2.1 adopting a binned model for NetFlow.

- **Cannot estimate the number of flows.** Large increases in the number of flows are telltale signs of denial of service attacks, scans, and worms which are much easier to notice when the traffic is measured in flows as opposed to bytes or packets [27]. Without help from the underlying protocols, it is impossible to recover the number of flows in the original traffic from the collected data [8]. Figure 3 illustrates the inherent error when trying to estimate the number of flows. Both traffic mixes have the same number of packets and go through the same sampling process. They produce similar flow caches: both have three entries, each with a packet counter of one. Any estimator working with these flow caches would give the same estimate for both traffic mixes, but in at least one of the cases the result will be significantly off since the second mix contains twice as many flows as the first one. We note here that the actual problem we want to solve is not counting the total number of flows but the related problem of counting the number of flows within specific aggregates (e.g., the number of SMTP flows, the number of flows coming from an IP address suspected of being a spam relay, etc.) There are good solutions for counting TCP flows since the first packet of each flow has the SYN flag set, which is recorded when present in the packets sampled by NetFlow [12] and our Adaptive NetFlow. Counting UDP and ICMP flows is equally important as these protocols are used for scanning, probing and spreading by worms such as Slammer [23] and Blaster and by malicious hackers. To solve this problem we propose in Section 3 the optional addition of new hardware that implements our Flow Counting Extension.

1.3 Related work

While sampling can be compensated for in reports that measure the traffic in packets or bytes it has been proven [8] that it is impossible to measure traffic in flows without bias. Duffield et al. [12] elegantly sidestep this impossibility result by using protocol level information present in the NetFlow records: they use the number of flow records with the TCP SYN flag set to accurately estimate TCP flows. We propose an optional flow counting extension which works for non-TCP traffic as well. In [13], Duffield et al develop estimators for flow length distributions and techniques for scaling measurements of sampled flow data. All of these techniques continue to be valuable and perform similarly under our Adaptive NetFlow.

There are solutions for counting the number of flows at line speeds [15], but these count the total number of flows, and one cannot later recover the number of flows associated with specific aggregates of traffic out of the overall mix. Our flow counting extension allows arbitrary aggregation of the flow keys in post-processing to obtain estimates of the flows in those aggregates.

Our flow counting extension is related to flow sampling introduced by Hohn and Veitch [18]. Important differences between FCE and their flow sampling are that we also outline a hardware implementation that can work at line speeds and that our primary focus is estimating the number of flows in arbitrary aggregates of traffic whereas theirs is computing the distribution of flow sizes. Choi et al. use adaptive sampling [9] to guarantee that the variance introduced by the variability of packet sizes does not exceed a pre-set limit.

There are solutions such as sFlow [26] that sample packets, but do not build flow records at all. The great advantage of these types of solutions is that they report full packet headers together with a portion of the packet payload and this provides much richer raw data for analysis. The disadvantage is that they do not benefit of the compression achieved by flow records that count more than one packet.

2. ADAPTIVE NETFLOW

Adaptive NetFlow is an improved version of NetFlow that addresses the shortcomings discussed in Section 1.2, mostly without changes to the router hardware or the infrastructure collecting traffic measurement data. Section 2.1 describes how we solve the mismatch between analysis and the flow termination heuristics of NetFlow by simply terminating flows only at the end of measurement bins. We eliminate the variability of the size of reported NetFlow data and solve the problems associated with the operator having to set the sampling rate by automatically adapting the sampling rate to the traffic mix (Section 2.2). When adapting the sampling rate ANF occasionally decreases the sampling rate to avoid filling all available memory. In order to keep the final results consistent with the new sampling rate, we need to adjust the packet and byte counters of the existing entries. We describe how we efficiently perform this "renormalization" in Section 2.2.1. If an entry's packet counter reaches zero during renormalization, meaning that none of its packets would have been sampled at the new sampling rate, that entry is deallocated. To ensure that ANF never runs out of memory, renormalization must free a certain number of entries. In Section 2.2.2 we describe the efficient and accurate method used by ANF to find the new sampling rate that ensures that the required number of entries are freed. In Section 2.3 we give an example of how a router manufacturer could configure ANF to ensure that *router memory and processing power are not exhausted under any possible traffic mix.*

Many proofs of the lemmas presented in this section are omitted for brevity. They can be found in the technical report version of this paper [14].

2.1 Operation in measurement bins

It is easiest to first address the mismatch between the NetFlow's flow termination heuristics and the analysis based on time bins used by applications. We can simplify processing of NetFlow data if all flow records reported fit within the bin used in analysis. We divide the NetFlow operation into short bins so that the bins used by traffic analysis are exact multiples of the measurement bins. We do not terminate flow records during the measurement bins, but terminate all active flow records at the end of the bin. To correctly place flow records in *analysis* bins, the analysis application needs to know only to which *measurement* bin it belongs. The timestamps of the first and last packet in flow records are no longer necessary, but they can be retained for backward compatibility. Alignment between measurement and analysis bins is not a problem if routers have accurate clocks, or use solutions such as the Network Time Protocol [22] to keep their clocks from drifting. Due to binning and to changes of sampling rate during the operation of ANF, the timestamps in the flow records cannot be used directly to derive flow length information. While flow length information might be derived indirectly, flow length statistics are outside the scope of this paper.

The size of measurement bins is a compromise between two opposing considerations. Larger measurement bins reduce the traffic generated by NetFlow since records are reported less often. On the other hand measurement bins must be at least as small as the smallest analysis bins, and smaller bins mean prompter reporting of network traffic. Choosing the bin size involves the same tradeoffs as choosing the active timeout that controls how long the entry of an active flow stays before being terminated and reported by Cisco's NetFlow to avoid staleness (default 30 minutes). Anecdotal evidence suggests that network operators often decrease the active timeout to obtain prompter reporting [2]. We conclude that the size of the measurement bin will have to be a

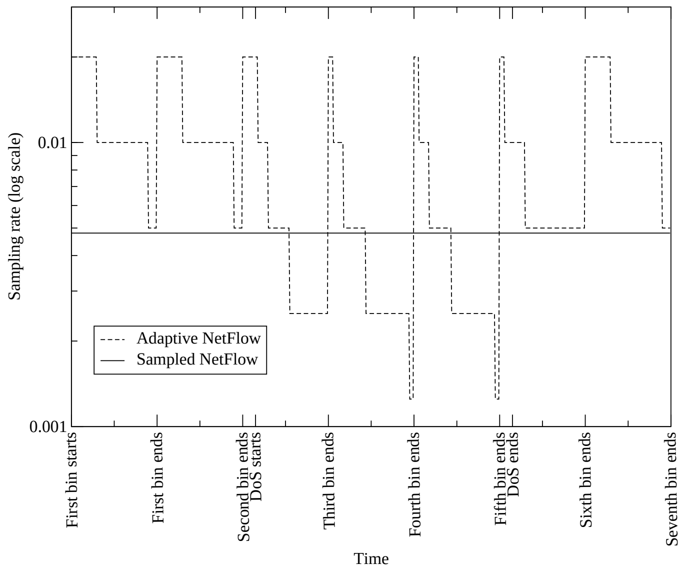

Figure 4: ***ANF automatically chooses a lower sampling rate during a DoS attack.*** **While NetFlow's sampling rate stays constant during a DoS attack, our Adaptive NetFlow switches to a lower sampling rate during each bin spanning the attack to keep the number of flow records generated constant. At the start of each new bin the rate is reset to the maximum, so that when the attack ends the rate is not kept unnecessarily low.**

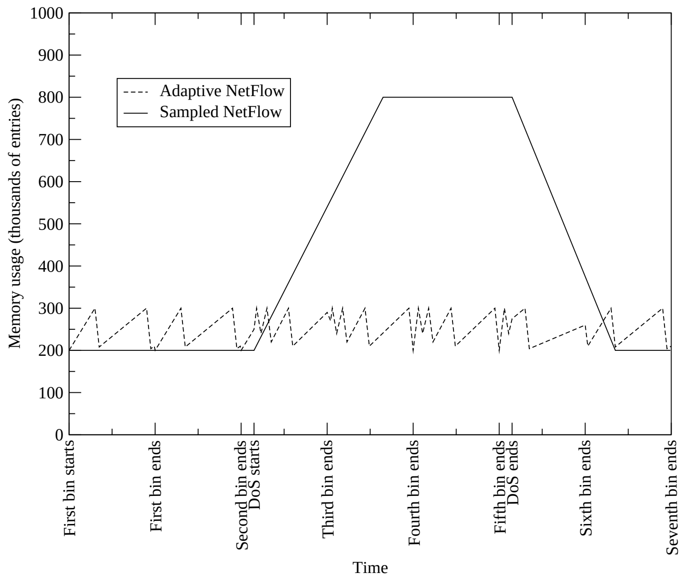

Figure 5: ***ANF limits memory usage during a DoS attack.*** **While NetFlow's memory consumption increases during a DoS attack our Adaptive NetFlow keeps its memory usage bounded.**

configurable parameter in Adaptive NetFlow. We consider that one minute and five minutes are both good choices for the default measurement bin size. In our experiments we used the more challenging one minute size for the measurement bins.

2.2 Adapting the sampling rate

Identifying the optimal sampling rate for NetFlow is hard because there are many conflicting factors to consider. One of them is avoiding overloading the processor that performs the NetFlow processing, whether it is the router CPU or a processor on a line card. The network operators must use trial and error to find the rate the router can support. Instead we propose that the router manufacturer determines the maximum sampling rate at which the processor can operate under worst case conditions. This traffic mix could occur for example due to a massive distributed denial of service attack.[1] We use this as the maximum sampling rate and we initialize the sampling rate to this value at the beginning of each bin.

Even if we start with a sampling rate low enough to not overwhelm the processor, the number of entries created can exceed the amount of available memory. With most traffic mixes, the number of entries created during a one minute measurement bin with the highest sampling rate the processor can support exceeds the tens to hundreds of megabytes of memory typically reserved for flow records. Instead of choosing the sampling rate in advance, ANF dynamically decreases the sampling rate until it is low enough for the flow records to fit into memory. Figure 4 shows how this process finds different sampling rates for normal traffic and for a DoS attack.

Traffic analysis multiplies the measured traffic by the inverse of the sampling rate to estimate the actual traffic, but if we keep changing the sampling rate while the flow records count the traffic, it is hard to determine what sampling rate to use as a basis for this compensation during analysis. To avoid this problem, we need to renormalize existing flow entries when we decrease the sampling rate. The renormalization process is equivalent to stopping the operation of NetFlow and going through all records to adjust the byte and packet counters to reflect the values they would have had if the new sampling rate had been in effect from the start of the bin. This way traffic analysis needs to know only the final sampling rate. Renormalization also removes the flow entries for which no packets would have been sampled with the new sampling rate. By freeing entries, renormalization ensures that there is enough memory to accommodate the records of new flows that appear until the end of the bin. Figure 5 illustrates how this effectively caps the memory usage during a DoS attack. The actual renormalization used by ANF does not require NetFlow to stop its operation, but operates concurrently. Section 2.2.1 gives a detailed description of how our efficient renormalization works and Section 2.2.2 explains how we use efficient and exact computations of the number of entries removed to find the new sampling rate that guarantees that renormalization removes enough entries to keep the memory from ever filling.

[1]Since this is a very unlikely scenario, especially for fast backbone links, in actual practice Adaptive NetFlow will never use more than a small percentage of the processor. Even if the router receives this unfavorable traffic mix due to a massive flooding attack, ANF will keep the processor fully loaded only for a short time until it decreases the sampling rate in response to the memory being consumed.

If the actual sampling rate is dynamic, one cannot ensure that it is the same for all routers nor even for different time bins at the same router. This does not cause problems with combining data from different bins or from multiple sources (e.g. combining the 60 one minute bins to get the traffic for the whole hour, or combining the traffic of many routers to get the traffic of a PoP) because we can simply add the counters from the flow records after having divided them by the respective sampling rates.

By adapting the sampling rate we ensure that ANF generates a fixed number of flow records. It is useful to quantify the accuracy of the analysis results one can obtain from a fixed number of records. Lemma 1 shows that, when estimating packets and bytes in arbitrary traffic aggregates that constitute a certain fraction of the total traffic, the worst case relative standard deviation depends only on the number of entries and not on the speed of the link. More simply put this means that you can slice and dice the data in any way, and as long as your slices are no smaller than a certain percentage of the pie, the relative errors of the estimates are small. For example, let's say we use a sampling rate that produces 100,000 flow entries, and network A accounts for 10% of the total packets, we will be able to measure its traffic with a relative standard deviation of at most 1%. If it accounts for 10% of the bytes, while the average packet size is 400 bytes and the maximum size 1500, we will be able to measure its traffic with an average relative standard deviation of at most 1.94% (irrespective of the size of the packets of network A).

LEMMA 1. *From the NetFlow records produced with independent random sampling at a rate at which the expected number of flow records is M, we can estimate the traffic of any aggregate amounting to a fraction f of the total traffic with a relative standard deviation of at most $\sqrt{1/(Mf)}$ for the number of packets and at most $\sqrt{s_{max}/(s_{avg}Mf)} < \sqrt{s_{max}/(s_{min}Mf)}$ for the number of bytes where s_{min}, s_{avg} and s_{max} are the minimum, average and maximum packet sizes.*

Proof Let T be the total number of packets sent during the measurement interval. With a sampling rate of M/T, the expected number of packets is M, and the expected number of entries is at most M. Thus the sampling rate at which the expected number of entries is M will be $p \geq M/T$. The number of packets in the aggregate is fT, and the number of those sampled has a binomial distribution with mean pfT and variance $p(1-p)fT$. Since we get the estimate for the number of packets in the aggregates by multiplying the number of sampled packets by $1/p$, the variance of the estimate will be $(1/p^2)p(1-p)fT = (1-p)fT/p < fT/p \leq fT/(M/T) = fT^2/M$. The standard deviation of this estimate will be at most $T\sqrt{f/M}$ and its relative standard deviation at most $T\sqrt{f/M}/fT = \sqrt{1/(Mf)}$.

When we measure traffic in bytes, the total traffic is Ts_{avg} and the traffic of the aggregate is $t_a \geq Ts_{avg}f$. The variance for the contribution of a packet of size s is $s^2p(1-p)$. Since packets are sampled independently, the variance for count for the entire aggregate is $\sum s_i^2 p(1-p) \leq \sum s_i s_{max} p(1-p) \leq s_{max}p \sum s_i = s_{max}pt_a$. Thus the variance of the estimate will be bound by $s_{max}t_a/p$ and its relative standard deviation by $\sqrt{s_{max}t_a/p}/t_a = \sqrt{s_{max}/(pt_a)} \leq \sqrt{s_{max}T/(Mt_a)} \leq \sqrt{s_{max}T/(MTs_{avg}f)} = \sqrt{s_{max}/(s_{avg}Mf)}$. ■

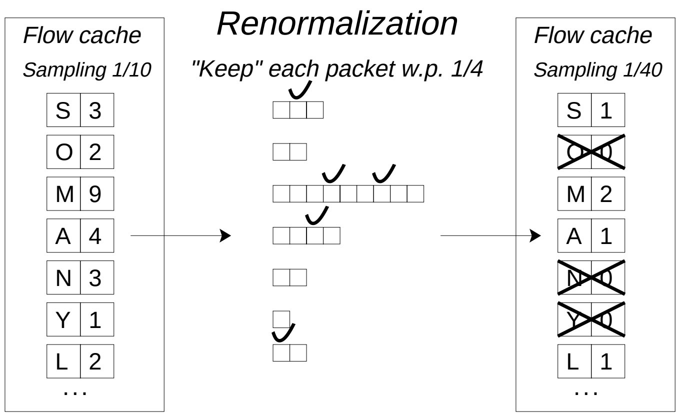

Figure 6: *Renormalization reduces packet counters and frees entries.* When our ANF reduces the sampling rate renormalization updates packet and byte counters (not shown) to values consistent with the new sampling rate. Entries whose packet counters reach 0 are freed. Notice how renormalization freed almost half the entries shown.

Our Adaptive NetFlow also alleviates the problem of exported flow records exceeding the available bandwidth. Since the number of flow records generated in each time bin is limited, the bandwidth consumed by sending them from the router to the collection station is also limited. By sending the flow records at a proper rate, the router can also ensure that the stream of measurement data is smooth. Recent discussions in IETF's IPFIX working group [6] advocate improving the reporting of traffic measurement data by splitting it into multiple levels and transmitting each with a different degree of reliability: when network conditions are favorable all data arrives at the collection station, but when the network is overloaded, just the most important level arrives intact, similar to receiver-driven layered multicast [20]. Adaptive NetFlow matches this approach well since our renormalization is the operation missing from NetFlow that can be used to generate progressively smaller traffic summaries to be sent using more reliable transport.

2.2.1 *Performing the renormalization*

The aim of adapting the sampling rate of ANF is to keep the number of entries generated under control. After starting the bin with a sampling rate that is safe for the processor, but not necessarily for the memory, ANF decreases the sampling rate whenever the number of entries allocated exceeds a certain threshold. After reducing the sampling rate we also need to renormalize the existing entries so that we bring the flow cache to a state consistent with having used the new sampling rate from the beginning of the bin, as shown in Figure 6. We consider that the packets that were not selected with the old sampling rate p_{old} would not have been selected with the new one p_{new} either, and the packets selected with the old sampling rate we keep with probability p_{new}/p_{old} and discard with probability $1 - p_{new}/p_{old}$.

One way to implement the renormalization for an entry with a packet counter of x would be to perform x random coin flips with probability p_{new}/p_{old} to compute the number of packets with the new sampling rate x_{new}. By noticing that x_{new} has a binomial distribution, we can more efficiently compute it with a single invocation of the random number generator. During renormalization, some entries will have their packet counters decrease to zero, and be removed from the flow cache. We also need to update byte counter y to reflect the new sampling rate and we update it proportionately to the change in the packet counters $y_{new} = y\ x_{new}/x$.

Implementing a binomial random variable involves significant amounts of computation. We use a simpler way of updating the packet count that maintains the expected value of x_{new}. We show in Lemma 2 that our simplified function for computing the new packet counter maintains the unbiasedness of the packet and byte counts and in Lemma 3 that and it actually reduces their variance. Let $r = p_{new}/p_{old}$ be the probability ratio. The expected value of x_{new} is $E[x_{new}] = rx$. If rx is an integer, our simplified update function sets x_{new} to rx, otherwise to $\lfloor rx \rfloor$ with probability $\lceil rx \rceil - rx$ and $\lceil rx \rceil$ with probability $rx - \lfloor rx \rfloor$ and this can be implemented with a single "coin flip". In the next section we show how to avoid even this one call per entry to the random number generator.

Lemma 2. *After any number of renormalizations using the simplified update function, the packet and byte counters for every flow stay unbiased.*

Lemma 3. *After any number of renormalizations using the simplified update function, the variance of the packet and byte counters for every flow is no larger than the variance they would have had the final sampling rate been in effect from the beginning of the interval.*

While conceptually the renormalization has to process all entries before we start processing packets sampled at the new rate, this is impractical because we would need a very large buffer to store the new packet headers while renormalization is in progress. In practice renormalization proceeds in parallel with the processing of new packets: for example after every 10 packets processed, we could renormalize 20 entries. Performing renormalization in batches that group entries located by each other in memory as opposed to after each processed packet improves the locality of memory references thus increasing the cache hit rate of the processor. All entries updated with packets sampled at the new sampling rate must be normalized before the update, so whenever we process a packet that maps to an entry that hasn't been renormalized yet we normalize it "on demand" before updating the packet and byte counters. To maintain a rate of two renormalization operations per packet processed, we change the rule from the example above to the following: after every 10 packets d of which triggered "on demand" renormalization, we perform a batch of $20 - d$ renormalizations.

To avoid falling behind, we must guarantee that the number of entries freed by renormalization is not smaller than the number of new entries created by packets sampled at the new sampling rate while renormalization is in progress. The number of packets sampled while renormalization is in progress is an upper bound on the number of new entries created. By setting an appropriately low initial sampling rate we can guarantee that the number of new entries created does not exceed a fixed number. But how can we guarantee that we clean a certain number of entries by renormalizing to a lower sampling rate?

The sampling probability for which renormalization frees a certain number of entries depends on the packet counters in the entries. Let's assume we want to reduce the number of entries to half. If all existing entries have a packet counter of one, reducing the sampling rate in half would eliminate approximately half of them. On the other hand if all entries had a packet counter of 3, decreasing the sampling rate to half would free no entries; we should decrease it by a factor of 6 to expect to clean half of the entries. In the next section we present our solution for finding the sampling rate for which renormalization frees exactly the desired number of entries.

2.2.2 Finding the right sampling rate

When we choose a new sampling rate for renormalizing, we need to ensure that a certain number of entries are deleted so that the new entries created by incoming traffic while renormalization is in progress don't push the size of the flow cache beyond the available memory. A full histogram of the sizes of the packet counters provides the information needed for finding the right sampling rate. With the simplified update function, the probability of an entry being removed is 0 if $rx \geq 1$ and $1 - rx$ otherwise where $r = p_{new}/p_{old}$. Using the histogram we can compute the expected number of entries to be freed during renormalization (n_i is the number of entries with packet counter i).

$$C = \sum_{i=1}^{i<1/r} n_i(1 - ri) \tag{1}$$

We can maintain the histogram by performing one more addition and one subtraction for each processed packet, to increment the flow counter of the histogram bin corresponding to the new value of the packet counter and decrement that of bin the old value belongs to. Keeping a histogram bin for every possible value of the packet counter can take significant amounts of memory. Fortunately the entries with packet counts higher than $1/r = p_{old}/p_{new}$ are never freed by our simplified update function. Therefore, to find the right sampling rate, we do not need the whole histogram, just the bins smaller than $1/r$. In the technical report [14] we show that we need at most $\lceil P/M \rceil$ histogram bins to renormalize so that the expected number of entries is M. It appears that the size of the histogram needs to increase linearly with the number of packets that can be sampled at the maximum sampling rate, imposing scalability problems. However the situation is better for the following 4 reasons:

- For typical configurations, the bound $\lceil P/M \rceil$ is not large – for example for the configuration discussed in Section 2.3 the bound is 187;
- The bound does not increase with the speed of the link, but with measurement bin size and processor speed;
- The traffic mix that actually requires the number of bins in the bound is very exotic: the link fully loaded with minimum size packets and with M flows equally sharing the bandwidth except for T-M flows with one packet sampled. In none of our experiments, including those running with initial sampling rate of 1 in 1 and with M eight times smaller than in the example from Section 2.3, did we need more than 5 histogram bins. Most often the first histogram bin (the one counting flow records with a packet count of 1) is

```
DECIDE_WHETHER_TO_REMOVE($crtentry$)
    $i = crtentry.packetcounter$
    $v_i$=$v_i + 1$
    if $r_i * v_i + s_i > f_i$
        $f_i = f_i + 1$
        return $true$
    else
        return $false$
    endif
```

Figure 7: *Calls to the random number generator not needed for each renormalized entry.* By associating a random seed and a small amount of other state with each histogram bin, we can perform the per entry processing using only integer arithmetic.

much larger than any of the others and together with the next few ones it accounts for more than 90% of the entries. DDoS attacks only increase this first bin because all sampled attack packets generate 1 packet flows. Therefore in practice setting the number of histogram bins to 32 is more than sufficient;

- Histograms with bin sizes increasing exponentially at a slow rate (e.g. each histogram bin is 10% larger than the previous one) give estimates for the number of bins freed with small bounded error and the number of bins they need is logarithmic in P/M. Since this solution is not necessary for configurations we consider realistic, we do not pursue it any further.

As discussed so far, renormalization requires a random decision for each entry, but calls to the random number generator are expensive. With periodic 1 in N sampling as an inspiration, we use the histogram bins to derive a more efficient method for updating the packet entries, without calls to the random number generator for each entry. For each histogram bin, we keep a counter for the number of entries in that bin visited for renormalization v_i a and the number of entries freed f_i. Let $r_i = 1 - ri$ be the probability of removing an entry with a packet counter of i and s_i a small random seed between 0 and 1 initialized before the start of the normalization. Figure 7 gives the criterion used at each entry to decide whether to remove it or not. Using the fact that all sampling probabilities are of the form $1/N$ where N is an integer, this decision can be implemented using only integer additions, multiplications, and comparisons. It is easy to extended efficient update to entries from other bins for which we need to decide which of two possible values to update the counter to. We can even extend it to the last histogram bin counting flows with 32 packets or more, but we also need to perform an integer division for entries from this bin. Since the expected value for each packet counter does not change with our efficient simplified update method, we introduce no bias. In the technical report [14] we also show that efficient update does not increase the variance of the estimates for aggregates.

Using the random seeds in the histogram bin, we can compute the exact number of entries freed in advance. This allows us to quickly find the exact sampling rate for which renormalization frees the desired number of entries. Can it happen that during normalization the number of freed entries is temporarily smaller than the number of new entries, due to the random order in which normalization processes the entries? It can, and exact modeling of this phenomenon is complex, but due to the fact that sampling with replacement has higher variance than sampling without replacement, we can upper bound the deviation by the deviation of a binomial process of T random decisions with probability M/T ($T > M$ is the threshold for starting renormalization). Thus leaving $3\sqrt{M(T-M)/T}$ extra entries is enough to accommodate these random variations with probability much larger than 99.87% even for the severest of DoS attacks.

2.3 Configuration example for Adaptive NetFlow

This section provides an example for conservatively configuring Adaptive NetFlow on an OC-48 interface (2488.32 Mbps). When choosing the configuration parameters, our aim is to ensure that the processor and the memory available for the flow cache are not overwhelmed under any possible traffic pattern. The work of finding the right values for the various parameters is the responsibility of the router manufacturer. All the router operator has to do is to specify the reported number of flow records M desired for each one minute measurement bin. If the memory on the line card is not sufficient, the router can override the parameter with a lower value. ANF guarantees that it will never report more flow records and it will report fewer only for pathological traffic mixes such as fewer than M large flows generating all the traffic or an extremely lightly loaded link.

The parameters the router manufacturer needs to determine are: the maximum sampling rate $p_0 = 1/N_0$, the average number of renormalization operations per processed packet when renormalization is in progress and the actual number of flow cache entries the router needs given the parameter M. By profiling the processor on the line card, the router manufacturer finds that the average time to process a sampled packet is $t_p = 3.4\mu s$ and renormalizing an entry takes on average $t_r = 1.5\mu s$.[2] We first compute the initial sampling rate and we want it as high as possible so that we get accurate results even on lightly loaded links. To keep up before the first renormalization, the per packet processing time for a link fully loaded with minimum size packets t_{min} could be as low as t_p, but we choose $t_{min} = t_p + t_r = 4.9\mu s$ to reduce the amount of memory needed. At a minimum size of 40 bytes for the IP packets to which we add 4 bytes of lower layer SONET CHDLC overhead, for our OC-48 we obtain a maximum sampling rate of 1 in 35 packets.

After the first renormalization starts we will process packets at a lower rate because the sampling rate decreases, but we will also need to perform the renormalization. If we set the renormalization threshold to $T = 2M$ entries, when we get there, the aim is to free M entries so we reduce the sampling rate to at least half (if we have entries with a packet count of more than 1 we will reduce it even more). The first renormalization is the critical one, as the others happen when lower sampling rates are in effect. During renormalization we have to process $2M$ entries and in the mean time we can receive up to M new packets so that the number of entries created does not exceed the number of entries cleared. With this configuration we need to renor-

[2]These are the actual values we obtained by profiling our code on a 750 MHz UltraSPARC-III which has performance comparable to that of a processor one would expect on an OC-48 line card.

malize two entries for every packet received. Since we decreased the sampling rate by at least a factor of two, we receive at most one packet every $9.8\mu s$. It takes $3.4\mu s$ to process it and $2 * 1.5\mu s$ for the normalization, so the processor is busy for $6.4\mu s$, so it can keep up. Since we actually have $3.4\mu s$ left for each packet we can set the threshold lower than $2M$ and still keep up during the first renormalization. By writing down the constraints we get a simple equation $(t_{min}T/M - t_p)(T - M) = Tt_r$. Its solution gives us the lowest threshold for which the first normalization can keep up with the rate at which the packets arrive: $T/M = (\alpha + \sqrt{\alpha^2 - 4t_p/t_{min}})/2$ where $\alpha = (t_r + t_p + t_{min})/t_{min}$. For our example we get $T = 1.56M$ and we need to perform $T/(T - M) = 2.8$ renormalizations for each sampled packet. To account for randomness in the entry clearing decisions we need to add $3\sqrt{M(T - M)/T} = 1.8\sqrt{M}$ entries, so the number of entries for our example is $1.56M + 1.8\sqrt{M}$.

ANF operates on time bins. At the end of the bin it needs to process the flow records collected during the bin to bring their number down to M and then to buffer them until they are transmitted to the collection station. Since our $t_{min} \geq t_p + t_r$, we can perform one renormalization on the old entries for each packet in the new bin, so by the time the number of entries in the new bin reaches T, we already renormalized all of the old entries, so we need only M more entries to support repeated bins. This brings our total to $2.56M + 1.8\sqrt{M}$ entries. For example if the network operator configures $M = 64K$ records per minute, this will generate a steady reporting traffic (NetFlow 7 fits 27 flow records into an 1500 byte packet) of 486 Kbps and it will use 168,232 entries that take 10.3 MB of router memory (NetFlow records use 64 bytes of memory). Note that reliable transfer of flow records as advocated by IETF's IPFIX workgroup requires buffering of records so that they can be retransmitted if packets are lost. The M entries for records from the previous bin we included in the memory needs of ANF are in fact a buffer and they can significantly reduce the amount of extra buffering needed in the router.

3. FLOW COUNTING EXTENSION

NetFlow entries record the SYN flag which is set in the first packet of each TCP connection. Using this information, it is possible to accurately estimate the number of active TCP flows in various aggregates (e.g. web traffic, traffic from Network A, etc.) even if NetFlow sees only a sample of the packets [12]. We retain this functionality in Adaptive NetFlow by setting the SYN flag in the flow entry when a SYN packet is sampled and maintaining it with probability x_{new}/x when renormalization decreases the packet counter of an entry from x to x_{new}. As with sampled NetFlow, Adaptive NetFlow data does not allow us to accurately estimate the number of non-TCP flows. To address this shortcoming we propose an optional addition, the Flow Counting Extension. FCE operates separately from ANF, and provides its own traffic measurement data whose only purpose is to support flow counts. FCE has the same high level properties as ANF: it can handle any traffic mix; it generates a constant amount of measurement data for each bin; it guarantees the relative standard deviation of the estimates for aggregates above a certain percentage of the total traffic; and the only configuration parameter the operator needs to set is the number of flow records reported per bin.

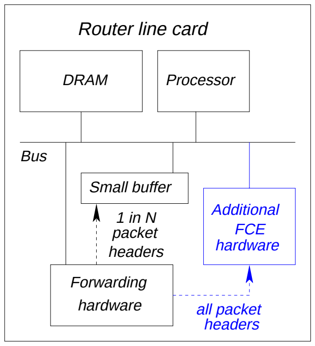

Figure 8: *The optional Flow Counting Extension requires additional hardware.* Since FCE needs to look at each packet header, to keep up with line speeds it must be implemented with additional hardware in high speed routers.

As a starting point for the development of FCE, we use an algorithm called "adaptive sampling", proposed by Wegman and described by Flajolet [17]. This algorithm solves the database problem equivalent to estimating the total number of flows, but we change it to support estimates for the number of flows of arbitrary aggregates ("slicing and dicing" the traffic). We keep a table with all flow identifiers seen in the traffic along with a hash of the flow ID. When the table fills, we make space for new entries by deleting the entries whose flow ID hash does not start with at least one 0 bit. Then, for each future packet, we insert its flow ID into the table only if its hash starts with one 0 bit (and it is not already in the table). When the table fills again, we keep only the entries whose hashed flow ID start with two 0s (increase the "depth" to 2) and so on. The output of the original algorithm is the estimate of the total number of flows, computed by multiplying the number of entries in the table by 2^{depth}. For example if we only keep the entries whose flow ID hash starts with two zeroes ($depth = 2$) we keep a quarter of all flow IDs, so if we report 1,000 web flows, we estimate that there were 4,000 web flows in the traffic mix.

We modify the adaptive sampling algorithm to output the current *depth* and the list of flow IDs in the table. To estimate the number of flows from a certain aggregate we just multiply the number of flows from that aggregate present in the output by 2^{depth}. The problem with this algorithm is that the number of entries it produces varies between 1 and 0.5 times the table size, and the accuracy of results varies accordingly. To ensure that we report close to M entries, we use a table of size $2M$, so that after each increase of *depth*, we have close to M entries. When the bin ends, if the number of entries is $L > M$, we perform an additional cleaning operation to bring the number of entries close to M. During a bin, the flows kept are those with flow ID hash $h < H/2^{depth}$, where H is the maximum hash value. In the end-of-bin cleaning pass we keep only flows with $h < H/2^{depth} * M/L$, and we report together with the flow IDs a correction factor of $N = 2^{depth}L/M$.

The function we use to hash flow IDs is a randomly generated member of the H3 hash function family[7]. Because of the randomness of this hash function, each flow appears in the output with a probability of $1/N$ and therefore the estimated sizes of the aggregates are unbiased. Assuming a perfect hash function, the sampling decisions for different flows are independent. We can use this to show that the expected number of flows in the output is M, with a small standard deviation of $\sqrt{M(1-M/L)} \leq \sqrt{M/2}$. We can also show that we can estimate the traffic of any aggregate amounting to a fraction f of the total number of flows with a relative standard deviation of very close to $\sqrt{1/(Mf)}$. Thus FCE provides the same types of guarantees for the accuracy of flow counts as ANF does for the accuracy of packet counts.

The Flow Counting Extension to Adaptive NetFlow is needed on high speed links, where we expect the traffic to be sampled because the processor cannot process each packet. At these speeds, since FCE needs to process each packet, we can implement it only using additional hardware (Figure 8). Computation of the hash function on the flow IDs can be implemented with combinatorial logic that is easy to pipeline. The table of flow IDs can be implemented with a CAM keyed on the flow IDs and the hash values. The CAM must support quick insertion and deletion of all entries matching a mask. We use this hardware primitive to implement cleaning of the CAM when we increase the *depth*. At the end of the interval, the flow IDs and their hash values are read from the CAM, and as the entries are read out the software performs the final cleaning to reduce the number of entries to close to M. After a bin is finished, the processor reads the entries of the CAM at slower than line speed, so we must provide additional CAM memory to allow recording flows of the new current bin while the processor works on the previous bin. A conservative solution is to use two CAMs of size $2M$ so that the second CAM can operate on the packets of the new bin while the first one is being read out.

We want to finally note here that the cleaning operation can be put to other uses too. Much like renormalization for ANF, additional cleaning operations performed in software at the router can produce smaller lists of flow IDs. Transmitting these progressively smaller summaries at progressively higher levels of reliability allows RLM-like [20] graceful degradation of traffic report accuracy in response to network congestion on the reporting path.

4. EXPERIMENTAL EVALUATION

In our experiments we use eight traces from various times of the day from OC-48 links at two different ISPs. The traces are summarized in the technical report [14].

4.1 Evaluation of Adaptive NetFlow

The aim of the experimental evaluation of ANF in this section is to compare the accuracy of its results with the theoretical bounds and with NetFlow.

Results from the technical report [14] show that as we increase the amount of memory the relative error of the aggregates decreases as expected. In Figure 9 we present the error in the estimates for all applications with more than 0.5% of the traffic, measured in packets, for report sizes of 8K and 256K entries. The plots show the 25th, 50th and 75th precentile over 25 runs. The byte estimates, omitted for brevity, display very similar trends. The first thing to notice is that the actual errors are generally below the errors predicted by Lemma 1, but more pronouncedly so for the large report sizes. The reason for this is that Lemma 1 assumes that all entries have a packet counter of 1, which is not true in either case, but the counters are lower for the small report size. It is interesting to note that there are some outliers such as NNTP for which the actual error is further from the theoretical bound than for other applications. The reason is the NNTP has the largest number of packets per flows over all applications and thus benefits most from our variance-reducing renormalization.

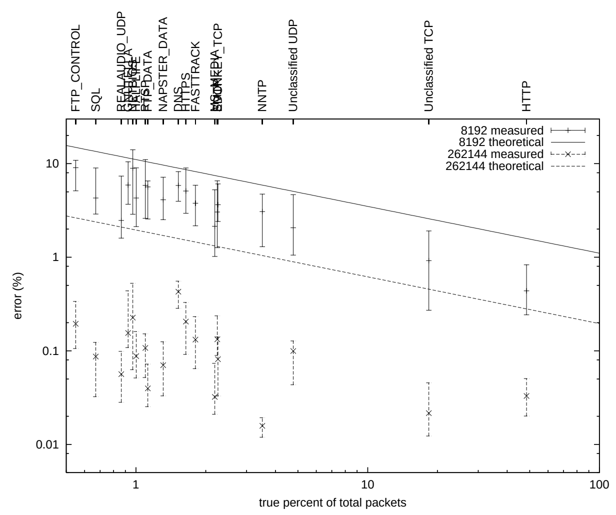

Figure 9: The error in estimating the number of packets for applications with differing amounts of traffic, with two different report sizes; vertical bars show the 25th, 50th, and 75th percentile of standard error over many runs. The straight lines show the theoretical standard error for the worst case scenario of all flows having only 1 packet; when run against a real traffic mix with larger flows, ANF produces errors which are even lower. This is dramatically illustrated by the very low error of estimates for NNTP, which has very large flows.

To verify that ANF's renormalization does not introduce bias or increase error, we compare ANF with a 64K entry report size to binned NetFlow configured statically with the sampling rate ANF stabilizes at by the end of the bin which is a sampling 1 in 57 (we call it "Psychic NetFlow" because it magically guesses the right sampling rate from the beginning of the bin). This type of NetFlow obviously could not work on a live link, as it requires *a priori* knowledge of the ideal sampling rate, but we can run it on recorded traces. Tables 2(a) and 2(b) show the results of one such comparison for 25 runs of each algorithm over a single trace bin. For both bytes and packets, these tables show that the renormalization in ANF did not introduce bias or increase the error.

4.2 Evaluation of the Flow Counting Extension

The experimental evaluation of FCE in this section compares the results of FCE with the theoretical bounds and estimators $\widehat{M_1}$ and $\widehat{M_2}$ based on counting the SYN flags from NetFlow records [13].

Table 3 shows that even for some TCP applications, the errors of FCE are much better than those for SYN based estimators, while for others the errors are similar. This reflects that the proportion of TCP flows without SYN flags or with duplicated SYN packets differs for different applica-

Aggregate	% of total	ANF bias	ANF st.dev	"Psychic NF" bias	"Psychic NF" st.dev
ALL Traffic	100	0.03	0.20	0.00	0.00
HTTP	48.	-0.02	0.30	-0.01	0.23
Unclassified TCP	18.	0.11	0.48	-0.00	0.55
ALL UDP Traffic	9.1	0.00	0.51	-0.02	0.81
AS 2914 src	7.6	0.21	0.64	-0.10	1.34
AS 2914 dst	4.9	-0.09	0.80	0.20	0.87
NNTP	3.5	0.09	0.73	0.00	1.38
SMTP	2.2	-0.07	1.59	0.97	2.11
HTTPS	1.6	-0.08	1.99	-0.70	2.30
DNS	1.5	0.34	2.10	-0.36	2.42

Table 2(a): Packet errors

Aggregate	% of total	ANF bias	ANF st.dev.	"Psychic NF" bias	"Psychic NF" st.dev
ALL Traffic	100	0.02	0.28	-0.02	0.32
HTTP	46.	0.02	0.31	-0.01	0.50
Unclassified TCP	24.	0.08	0.64	0.04	0.79
AS 2914 src	10.	0.20	0.72	-0.08	1.76
NNTP	6.4	0.08	0.80	-0.33	1.65
ALL UDP Traffic	3.6	0.00	0.79	-0.13	1.60
AS 2914 dst	3.6	-0.13	1.03	0.44	1.68
SMTP	1.3	-0.28	2.11	1.11	4.31
HTTPS	0.8	-0.18	2.77	-0.38	3.52
DNS	0.3	-0.05	2.36	-0.30	3.38

Table 2(b): Byte errors

Table 2: The bias and standard deviation of errors for various applications as produced by ANF are not much different than those of "Psychic NetFlow".

tions. And FCE does equally well on aggregates containing non-TCP flows, where SYN-based estimation does not work.

4.3 Performance under extreme traffic mixes

Finally we show that the memory and bandwidth usage of ANF and FCE are essentially unaffected by extreme traffic mixes by testing with a simulated denial of service attack. We mix from 16 thousand to 6 million packets per second (1 to 360 million packets per minute) into a trace. The December 2003 attack on SCO was 1 million packets per second. The results presented in the technical report [14] confirm that the memory and bandwidth usage stay constant for ANF while increasing significantly for NetFlow.

Figure 10 shows how the errors for the estimates of certain aggregates change with various levels of DoS for a fixed configuration of ANF. While the relative errors for most applications have an increasing trend this does not contradict our theoretical results, because as the size of the attack increases, the legitimate traffic represents a smaller and smaller percentage of the total traffic.

5. CONCLUSIONS

NetFlow is the traffic measurement solution most widely used by ISPs to determine the composition of the traffic mix on network links. However, NetFlow has several important problems that we address with the improvements proposed in this paper. We make no claim to have exhausted all the opportunities for improvement and we express confidence that the networking community will soon finds further ways of advancing the state of flow level traffic measurement.

Our Adaptive NetFlow, deployable by a simple software update to routers, achieves robustness lacking in NetFlow by

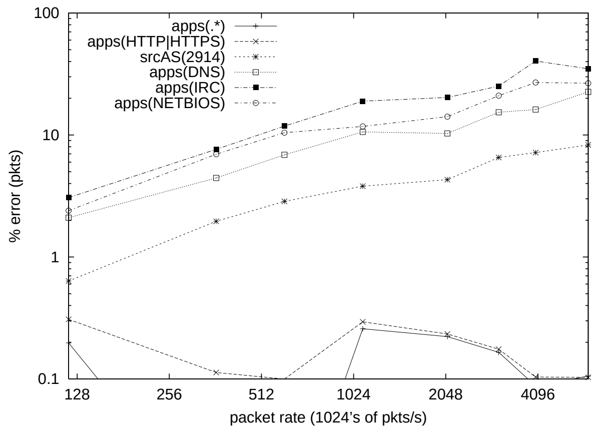

Figure 10: Error for the estimates for the number of packets sent by aggregates as DDoS increases.

adapting the sampling rate to the traffic mix. ANF provably stays within a fixed memory and reporting bandwidth budget for all possible traffic mixes. Our fast renormalization method is a key component of Adaptive NetFlow that allows us to also guarantee that the processor performing NetFlow at the router can keep up with any traffic mix. Further advantages of ANF over NetFlow are easy configuration as the network operator does not need to statically set the sampling rate, but only the rate at which flow records are produced, and simpler and more accurate analysis because we match the time bin model used in traffic analysis.

From sampled NetFlow and ANF data one can give accurate flow counts for the TCP flows in the traffic based on the SYN flags recorded in the flow entries. To enable consistently accurate counts for non-TCP flows, we propose the optional Flow Counting Extension that requires the addition of new hardware to high speed routers.

An important new feature of both ANF and FCE is that they can easily provide progressively smaller but less accurate summaries which can be transmitted to the collection station with progressively higher levels of reliability as discussed by IETF's IPFIX working group. For both ANF and FCE we present theoretical analyses that upper bound the relative error of the estimates produced for large aggregates measured in packets, bytes and respectively flows. We also prove that these estimates are unbiased. Measurements on multiple traces of traffic confirm our theoretical analysis. Measurements on traces with synthetic DoS attacks of various sizes confirm the robustness of our solutions.

6. ACKNOWLEDGEMENTS

Support for this work was provided by NSF Grant ANI-0137102, DARPA FTN Contract N66001-01-1-8933, and CAIDA members.

7. REFERENCES

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

[2] Personal conversation with Dave Plonka.

Aggregate	% of total	FCE		$\widehat{M_1}$		$\widehat{M_2}$	
		bias	st.dev.	bias	st.dev.	bias	st.dev.
ALL Traffic (*)	100	0.02	0.96	-35.55	35.55	-25.57	25.58
ALL TCP Traffic	78.	0.10	1.16	-17.39	17.41	-5.78	5.83
HTTP	58.	0.27	1.29	-19.24	19.26	-8.50	8.54
ALL UDP Traffic (*)	20.	-0.13	2.26	-100.00	100.00	-96.01	96.01
DNS (*)	8.0	0.03	3.94	-99.26	99.26	-95.31	95.31
Netbios (*)	7.9	-1.97	3.90	-39.27	39.35	-37.37	37.45
AS 2914 src (*)	7.2	0.92	5.43	-15.66	16.06	-5.70	6.69
Unclassified TCP	5.1	2.19	5.60	-47.07	47.17	-27.43	27.59
SMTP	2.3	-0.54	5.96	0.56	5.74	13.50	14.52
ALL ICMP Traffic (*)	1.5	-2.12	8.54	-100.00	100.00	-95.45	95.45
POP	0.3	4.23	19.01	17.71	26.85	32.35	38.17
IRC (*)	0.3	-9.01	18.32	-71.48	71.94	-56.20	56.80

Table 3: The flow count errors and biases in trace1 for various applications. The report sizes are 8K entries for FCE and 64K entries for $\widehat{M_1}$ and $\widehat{M_2}$. Aggregates marked with (*) may contain non-TCP flows.

[3] Paul Barford, Jeffrey Kline, David Plonka, and Amos Ron. A signal analysis of network traffic anomalies. In *Internet Measurement Workshop*, November 2002.

[4] Andy Bierman and Juergen Quittek. Packet sampling (psamp). IETF working group.

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

[6] Nevil Brownlee and Dave Plonka. IP flow information export (ipfix). IETF working group.

[7] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. In *Journal of Computer and System Sciences*, volume 18, April 1979.

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

[9] Baek-Young Choi, Jaesung Park, and Zhi-Li Zhang. Adaptive random sampling for load change detection. In *SIGMETRICS*, 2002. (extended abstract).

[10] Chuck Cranor, Theodore Johnson, Oliver Spatschek, and Vladislav Shkapenyuk. Gigascope: A stream database for network applications. In *Proceedings of the ACM SIGMOD*, June 2003.

[11] Nick Duffield and Carsten Lund. Predicting resource usage and estimation accuracy in an IP flow measurement collection infrastructure. In *Internet Measurement Conference*, October 2003.

[12] Nick Duffield, Carsten Lund, and Mikkel Thorup. Properties and prediction of flow statistics from sampled packet streams. In *SIGCOMM Internet Measurement Workshop*, November 2002.

[13] Nick Duffield, Carsten Lund, and Mikkel Thorup. Estimating flow distributions from sampled flow statistics. In *Proceedings of the ACM SIGCOMM*, August 2003.

[14] Cristian Estan, Ken Keys, David Moore, and George Varghese. Building a better NetFlow: Technical report, 2004. `http://www.caida.org/outreach/papers/2004/tr-2004-03/`.

[15] Cristian Estan, George Varghese, and Mike Fisk. Bitmap algorithms for counting active flows on high speed links. In *Internet Measurement Conference*, October 2003.

[16] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, Jennifer Rexford, and Fred True. Deriving traffic demands for operational IP networks: Methodology and experience. In *Proceedings of the ACM SIGCOMM*, pages 257–270, August 2000.

[17] Philippe Flajolet. On adaptive sampling. *COMPUTG: Computing (Archive for Informatics and Numerical Computation), Springer-Verlag*, 43, 1990.

[18] Nicolas Hohn and Darryl Veitch. Inverting sampled traffic. In *Internet Measurement Conference*, 2003.

[19] Ken Keys, David Moore, Ryan Koga, Edouard Lagache, Michael Tesch, and k claffy. The architecture of CoralReef: an Internet traffic monitoring software suite. In *PAM2001*. CAIDA, RIPE NCC, April 2001. `http://www.caida.org/outreach/papers/2001/CoralArch/`.

[20] Steven McCanne, Van Jacobson, and Martin Vetterli. Receiver-driven layered multicast. In *ACM SIGCOMM*, volume 26,4, pages 117–130, New York, August 1996. ACM Press.

[21] Keith McCloghrie and Marshall T. Rose. RFC 1213, March 1991.

[22] David L. Mills. RFC 1305: Network time protocol (version 3) specification, implementation, March 1992.

[23] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. The spread of the sapphire/slammer worm. Technical report, 2003.

[24] Cisco netflow. `http://www.cisco.com/warp/public/732/Tech/netflow`.

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

[26] Peter Phaal, Sonia Panchen, and Neil McKee. RFC 3176: sFlow, September 2001.

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

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

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