A Comparative Experimental Study of Hash Functions Applied to Packet Sampling

M. Molina[A] and S. Niccolini[B] and N. G. Duffield[C]

[A] DANTE, City House 126-130 Hills Road, Cambridge CB2 1PQ, United Kingdom
email: maurizio.molina@dante.org.uk
[B] NEC Europe Ltd., Kurfürsten-Anlage 36, 69115 Heidelberg, Germany
email: saverio.niccolini@netlab.nec.de
[C] AT&T Labs-Research, 180 Park Avenue, Florham Park, NJ 07932, USA
email: duffield@research.att.com

Abstract: Traffic measurements in high-speed links are challenging and risk introducing prohibitive costs in the monitoring infrastructure. Packet sampling is an established technique to make these network measurements feasible. Hash based packet sampling is a technique that, in addition, enables new applications, like trajectory sampling and One Way Delay estimation, provided that the same hash function is used in all measurement points. Such applications require that the used hash functions have specific properties: uniformity of distribution, computation speed, and low collision probability. In this paper we describe a methodology for hash function comparison and related test results, which have been contributed to the IETF PSAMP (Packet SAMPling) working group.

Keywords: Network monitoring, Traffic measurements, Hash functions, Packet sampling

1. INTRODUCTION

Managing today's complex Internet requires increasingly detailed traffic measurements [3]. These are essential inputs for several applications, including network planning, traffic engineering, verification of performance targets in Service Level Agreements (SLAs), detection of hot spots and denial of service attacks, and usage-based accounting. Measurement techniques can be broadly divided in two categories: active and passive. In active measurement, probe traffic is injected into the network, usually for the purpose of measuring the network performance experienced by the probes. Passive measurement entails measuring traffic already present in the network. The different types of measurement, active and passive, use different measurement techniques and support different, although not disjoint, sets of applications. We will not discuss the relative merits of the different types (see e.g. [7]) but concentrate on passive measurements only. In order to support the applications mentioned at the start, it is no longer sufficient to collect aggregate measurements, such as interface counters available ubiquitously from routers through SNMP. Supporting the data needs of the applications mentioned above requires measurements differentiated down to the level of individual routes, networks, customers and host applications. In order to support arbitrary "slicing and dicing" of the traffic for analysis and measurement applications, it has become necessary to collect measurements at the finest granularity, i.e. individual packets or flows.

The main challenge here for passive measurement is that it is not generally possible to measure and report on all individual packets flowing through Measurement Points (MPs), because this would lead to a

monitoring infrastructure with high hardware requirements and costs [1]. A well-established technique for reducing the cost of a passive measurement infrastructure is to sample only a subset of the flowing traffic. Statistical estimation must then be applied in order to infer the required properties of the original traffic stream and to determine the reliability of the estimates. As a very simple example, suppose in a certain time interval T, that 1 out of N packets are sampled on average (e.g. independently or periodically), and that the goal is to evaluate the number X of packets flowing within such an interval starting from the number x of sampled packets within this interval. Then, an unbiased estimate of X is simply xN, and the variance of this estimate (assuming that the samples are uncorrelated) is proportional to $1/x$ [1]. More sophisticated sampling techniques are possible. A detailed description of them can be found in [9], a document of the IETF PSAMP Working Group [8]. PSAMP has the mandate to standardize the way to perform packet sampling and to export sampled packets. The ultimate goal is to ease the deployment of a large (multi domain) passive monitoring infrastructures based on sampling, where results collected in different portions of the network (potentially by equipment of different vendors) can be compared and correlated because they are based on the same standard sampling techniques [9]. Hash-based packet selection is one of the new sampling techniques described in [9]. Each router computes a hash over fields of a packet that are invariant along its path, and select the packet for measurement if the hash falls in a given range. If a set of MPs employs a common hash function for sampling, each packet is selected either by all MPs that it encounters or by none. This fundamental property enables a powerful set of new measurement applications that we will detail in Section 2. In order to fulfill the aim of ubiquitous support for hash-based sampling, the hash must be sufficiently simple to be computable at line rate. On the other hand, the hash must be sufficiently strong that packet selection has good statistical properties. These two properties are in opposition. The purpose of this paper is to establish a methodology for determining which candidate hash functions strike the correct balance between strength and computability. In particular, we evaluate the hash functions experimentally, based upon their performance when applied to traffic traces. The broader aim of this work is to provide a scientific basis for the choice of hash functions for the emerging PSAMP standard. We feel that this is an important endeavor, since the hash function will likely, in some implementations, be encoded in hardware and hence will have a relatively long development time. Neither these implementations nor the PSAMP standard will be lightly or easily changed. Thus is it important to make the best choice of hash function at the outset. The outline for the rest of the paper is as follows. In section 2 we describe what hash-based selection is and the monitoring applications that it enables. In section 3 we state the requirements of the hash functions and describe a methodology for their comparison. Results of this comparison are in section 4. Section 5 concludes the paper and outlines future work.

2. HASH BASED PACKET SELECTION AND ITS APPLICATIONS

There are several possible ways to sample from a stream of packets. Perhaps the simplest is periodic sampling of the N^{th} packet, the $2N^{th}$ packet and so on. This can be implemented by decrementing a counter when a packet arrives, selecting a packet when the counter reaches zero, then resetting the counter to N. Or packets may be selected independently with probability $1/N$ using a pseudorandom number generator. To sample at a given time frequency, one can simply select the first packet after a looping timer expires. The performance of classical packet sampling techniques has been studied in [2]. All these techniques are suitable if the monitoring application works with samples coming from a *single* Measurement Point. But if the application needs to relate packets coming from *different* MPs, one must ensure that the same set of packets is selected at each MPs; we call this *consistent selection.* Clearly, sampling packets independently across different MPs is not effective; since the probability of a given packet being selected at all of m routers is only N^{-m}. The main application of consistent selection is trajectory sampling, where a central

Data Collector (DC) can understand the path of a selected packet in a complex network by collating the reports coming from the MPs it crosses (Figure 1). Trajectory sampling has been proposed in [5] as a monitoring method that enables new network management applications or enhances existing ones.

- *Network Engineering*: the fundamental new data provided is the *path matrix*. This describes not only the intensities of traffic between origins and destinations (i.e. the OD traffic matrix) but the intensities along the specific network path that traffic between a given OD pair follows. The path matrix is measured directly, without the need to track or join with network-wide routing information, as other monitoring methodologies require [16].
- *Real-time Route Troubleshooting*: routing loops are manifest as self-intersecting trajectories. The transient effects of routing and other path changes are measured directly, without the need for up to date routing data, or assumptions concerning routing stability and convergence times. The time required for identification is limited only by the latency in reports reaching the DC, and in the grouping and analysis of reports on a given packet.
- *Passive Performance Measurement*: loss packets are manifest in incomplete trajectories. If packet reports also contain a timestamp, and the MPs are accurately synchronized (e.g. via low cost GPS receiver, see [17]) the DC can also estimate the packet's transit delay between the MPs (One Way Delay). See Figure 1.

In this paper, we assume consistent selection is realized through hash-based selection. Other realizations (e.g. selection on the basis of a packet mark randomly assigned at the network ingress) are conceivable [18], but have also drawbacks; see [3] for further discussion. Our hashing terminology is contained in the following passage from [9]: "A hash function h maps the packet content c, or some portion of it, onto a hash range R. The packet is selected if h(c) is an element of S, which is a subset of R called the hash selection range". The portion of the packet content over which the hash function operates is called the "hash key", or simply the "key". The hash range is the set of values the hash function can take. For example, if the hash function's output is a 32 bit string, the hash range is $[0, 2^{32}-1]$.

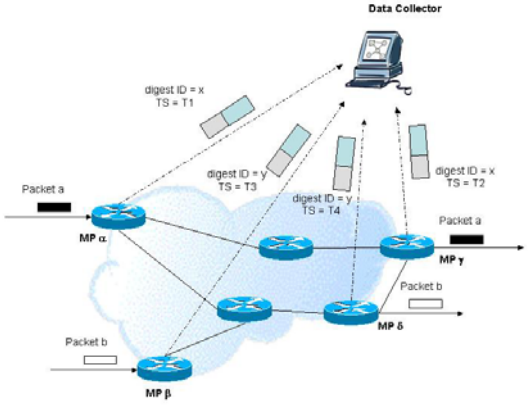

Figure 1. Trajectory sampling and One Way Delay estimation: selected packets generate reports to a DC

p	P (wrong_OWD)	
	RT=10^5	RT=10^4
1.E-06	6.34E-02	6.63E-03
1.E-07	6.63E-03	6.66E-04
1.E-08	6.66E-04	6.67E-05
1.E-09	6.67E-05	6.67E-06
1.E-10	6.67E-06	6.67E-07

Table 1. Probability of a OWD report pair suffering a digest ID collision

For consistent packet selection the key must be common between at all MPs. For example, for an IP packet with no Network Address Translators (NATs) between the MPs, all the path-invariant fields of the IP header and all the bytes of the packet payload can be used as packet keys. Fields that can vary from hop to hop, like the Time-to-Live (TTL), the Cyclic Redundancy (CRC) or the Differentiated Services Code Point (DSCP) must be excluded. See [5] for a complete description of the IP header fields suitable to be part of the key. As shown in Figure 1, each selected packet generates then a report (possibly, containing a

timestamp) that is sent to a DC. The digest ID is a bit string that allows the DC to relate several reports of the same packet. The digest ID can be the full packet itself, or a portion of it. But if the collector application is not interested in the packet content (as e.g. in the case of One Way Delay estimation) for saving bandwidth it is better to compute the digest ID through another hash function and send therefore a more compact bit string. In this case, as [5] explains, it is fundamental that the hash function used to compute the digest ID is independent from the one used for packet selection. Issues arising when collating packet reports at the DC, e.g. management of timeouts, the efficient representation of trajectories, the management of collisions in a hash-based digest ID (i.e. when distinct packets hash to the same digest), and the impact and management of loss of reports on applications, are described in [4], [6] [11]. If the portion of the packet composing the key is sufficiently variable across the packet stream, and the hash function sufficiently strong (in the sense that nearby keys have widely different hashes) then hash-based sampling ensures not only consistent packet sampling along the path, but also that the selection will approximate random and independent sampling across the packet stream. Therefore, hash based sampling can emulate probabilistic sampling without the need of using random probability generators.

3. REQUIREMENTS OF HASH FUNCTIONS AND METHODOLOGY FOR HASH FUNCTION COMPARISON

To incorporate hash-based selection into Internet standards, the IETF PSAMP WG must give a precise indication of which hash functions must be available on MP equipment. Therefore, we made a performance comparison of several hash function and contributed the results to PSAMP WG. In the previous section we explained that two independent hash functions are needed, one for hash-based packet selection and (optionally) one for digest ID generation. Before presenting the results, we must clarify the comparison criteria we used in both cases.

3.1. Hash Function Requirements for Packet Selection

The hash function used for packet selection has two fundamental requirements:

- it must be computed quickly, at line rate
- the output must be sufficiently uniform on the hash range

With uniformity, the sampling ratio is size(S)/size(R), where R and S are the hash range and the selection range, independently of how S is chosen. Suppose for example that the hash range R is [0,65535]. Then if the distribution is uniform, it is equivalent to define S as [0, 32767] or [32768, 65535] in both cases the sampling ratio will be 50%. This property is important for security: even if the hash function is public, keeping S configurable (and secret) prevents the crafting of packets to evade selection [9].

3.2. Evaluation Methodology for Hash Function Speed

The metric we used for evaluating the hash function speed was trivially the execution time of a single hash computation, averaged over ten millions of repetitions. We used a Dell notebook with a Pentium 4m, 1.7Ghz (512Kbytes cache) running Linux OS. We tested software implementations of the hash functions, being as “fair” as possible, e.g. passing the same type and number of parameters, avoiding sub-calls etc.

3.3. Evaluation Methodology for Hash Uniformity

We can evaluate the uniformity of the distribution of hash values in two ways. Firstly, we can ask whether the distribution is statistically distinguishable from the uniform distribution. But even when the distributions are distinguishable, they may not differ to an extent that concerns us in practice. A second approach is then to formulate a metric that more directly expresses the closeness of the distributions. In our comparisons we partition the hash range into N equal size bins. Suppose we calculate the hashes of P keys. Under the uniform hash distribution, the expected number of hashes falling into each bin is $f_0 = P/N$.

The actual number of hashes falling into bin i will be denoted by f_i , $i = 1,\ldots,N$.

3.3.1 Significance Test Approach

Conformance of an empirical distribution with a model can be evaluated with the Chi-Square test. This employs the statistic:

$$\chi^2 = \Sigma_{i=1,\ldots,N} (f_i - f_0)^2/f_0 \quad (1)$$

The quantity χ^2 acts as a measure of deviation of the f_i from the uniform distribution. Under the *null hypothesis* that the hash keys are independently and uniformly distributed, χ^2 follows a chi-squared distribution with N degrees of freedom. The so called P-value α, i.e. the probability for χ^2 not to exceed the value actually taken, can be calculated from tables. In standard practice, if α is larger than a specified value α_0, known as the *confidence level*, the null hypothesis is rejected. This is saying that the obtained value of χ^2 is too rare for the null hypothesis to be plausible. Rather than depending on a single hypothesis test, in this paper we conduct 60 independent tests, each of which computes hashes from P = 400,000 keys. At a confidence level α_0 the null hypothesis is correctly accepted with probability α_0. In other words, the average proportion of the experiments in which α is smaller than any value α_0 is just α_0. Hence the values of α obtained in the 60 experiments, when plotted in increasing order, would lie on a straight line of slope 1/60, on average. We will use these plots to compare conformance with the model across different hashing functions.

3.3.2 Variability Metric Approach

Whatever strong the hash function is, the distribution of hashes obtained from a set of (non-uniform) real keys should be statistically distinguishable from the uniform distribution, since even small non-uniformities in the hash will persist for large set of keys. But small non-uniformities may not affect measurement applications, even when statistically detectable with the formal χ^2 test. We capture these non-uniformities in the variability metric:

$$V = N*\{STD(avg\_f) + \tfrac{1}{2} AVG(cfi\_95)\} \quad (2)$$

Here, avg_f denotes the average and cfi_95 the 95% confidence interval of a frequency f_i over all 60 experiments, while STD and AVG denote their standard deviation and average over the N bins. The first term in (2) captures consistent differences across bins of the frequencies f_i, the second term captures variability across different experiments.

3.4. Hash Function Requirements for Digest ID Generation

The hash function used for digest ID generation has one fundamental requirement, which is to have a low collision probability of digest IDs over timescales relevant for applications. Note that even under a "perfect" hashing assumption that hash values are independent and identically distributed over a digest of m bits, the hashes of two distinct packets will collide with probability $p = 1/2^m$. In practice the collision rate may be higher than this due to (i) non-uniformity of the hash distribution; and (ii) distinct packets having the same hash key for digesting. The uniform case allows use to establish a baseline for actual collision frequencies. To be specific, consider the case of One Way Delay (OWD) estimation. In this application, the DC groups reports generated from different MPs by matching the digest ID. In [11] the expected proportion of reports pairs from the same packet that suffer a digest collision with a report on another packet that prevents correct estimation of OWD is bounded above by:

$$P(\text{wrong\_OWD}) = (2/3)\,[1-(1-p)^{RT}] \quad (3)$$

Here, T is an upper bound on the time between receipt of reports on the same packet at the DC, and R an upper bound on the rate at which reports reach the DC. Hence, RT is an upper bound for the maximum number of packets that reach the collector between reports of a given packet, and hence $1-(1-p)^{RT}$ is an upper bound for the probability that a given packet has a hash collision. The factor 2/3 arises since not all orderings of report arrivals from packets with colliding digests actually give rise to an incorrect

determination of the OWD. Using reasonable values for R and T, it can be seen (Table 1) that if we want for example this probability to be lower than 10^{-4}, (i.e. shaded cells in the table) collision probabilities as low as 10^{-9} may be needed for some RT combinations, corresponding to a digest length m of at least 30 bits. Computation speed is still a requirement for the hash function used for digest ID generation, but this requirement is less stringent than with the hash function used for packet selection, because the digest ID must be computed only on the packets that are selected during sampling (see Figure 1). We comment upon how collisions are to be identified, and the actions that should be taken upon their discovery. When reporting to the DC is reliable (e.g. by export over a dedicated management network, or by using a reliable transport protocol such as TCP for export), hash collisions can be identified by the presence of collisions in hashes reported from MPs located at the network ingress points. In this case, all colliding reports may be discarded. Discard is independent of packet path, and hence no bias is introduced into estimators based on the surviving samples. If export to the DC is unreliable, it is still possible to identify duplicates if MPs at ingress provide additional information on the set of packets selected; see [6].

4. COMPARISON RESULTS

4.1. Brief Description of Tested Hash Functions

For the hash function for packet selection we considered four candidates: CRC32 [12], IPSX [9], MMH [13], and Bob [14]. CRC32 is the standard Cyclic Redundancy Check (CRC) hash function. IPSX (IP Shift and XOR) is a simple hash function doing only XOR and shift operations: it operates on keys of fixed length (16 bytes) and is tailored for IPv4 packets. MMH (Multi-linear Modular Hash) uses a scalar vector multiplication and approximates a modulus prime operation, without doing any real computing-intensive modulo operation. The Bob hash does mostly the same type of operations as IPSX (shifts an XOR), but repeats them more times and also includes additions and subtractions. All functions can operate on keys of any length, except IPSX, which is currently defined only for 16 byte keys. The hashes were selected for evaluation based on perceived strengths: CRC32 for established use the IP protocol, IPSX for its simplicity. MMH and Bob were the best hash functions in a preliminary screening performed on wider set [15]. All the functions can give an output on 32 bits. For digest ID generation we keep all 32 bits to keep the collision probability low. For packet selection purposes the output can be masked, because the minimum sampling ratio is likely be much higher than $1/2^{32}$. For example, in out tests we considered 1/10,000 as the minimum sampling ratio.

4.2. Hash Functions for Packet Selection – Uniformity Tests

We performed two sets of tests, one with synthetically generated keys, i.e. where the keys are a sequence of L 32 bit words each one randomly chosen between $[0,2^{32}-1]$. The synthetic trace represents an "ideal" situation where the input keys are already uniformly distributed. In the second test, the keys comprised L words extracted from packets of a real trace collected in NEC-Europe labs in Heidelberg. As IPSX operates on keys of fixed length of 16 bytes, we always used L=4, also for the other hash functions that would not be constrained to a fixed key size. In the real trace case we selected as keys the following fields of the IPv4 packet header: Total length, Identification, Source IP and Destination IP fields (three words in total and the 2^{nd} 32 bit word of the packet payload (which, if the packet is UDP, contains the UDP length and checksum fields; if it is TCP contains the TCP sequence number). This choice is motivated by the fact that all the fields are path-invariant, thus allowing consistent packet selection (Section 2), but at the same time potentially highly variant from one packet to another [5], and this helps to approximate independent sampling. Finally, our experiments used N=1,000 and N=10,000 bins; we mostly report only the first case due to space limitations.

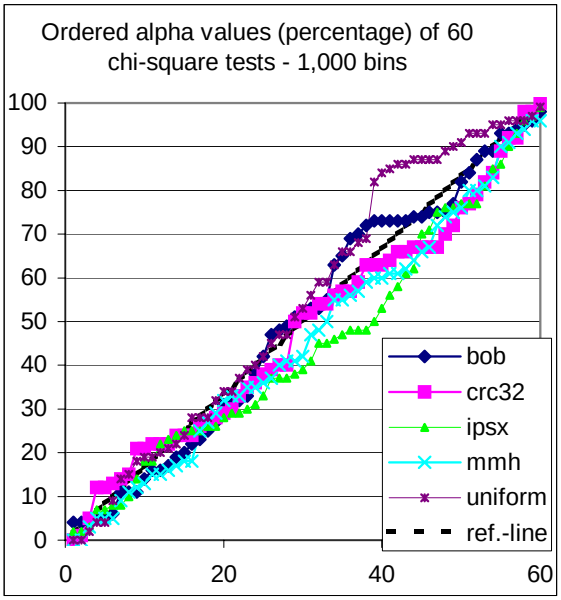

Figure 2. Testing uniformity with χ^2 test, synthetic trace

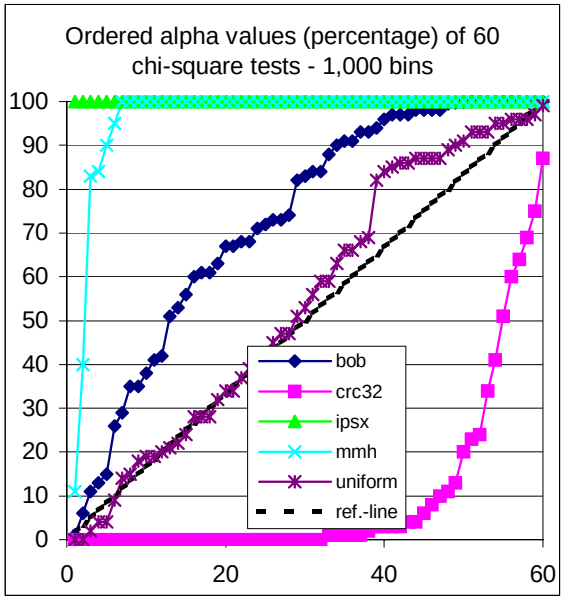

Figure 3. Testing uniformity with χ^2 test, real trace

4.2.1 Results for Significance Test Approach

Graphs of ordered α values are shown in Figure 2 (for the synthetic trace) and Figure 3 (for the real trace). The dashed, straight line is the reference average behavior for the uniform distribution. Drifts *above* this reference indicate a "bad" uniformity, *below* it an uniformity "even better than what expected". We also plot the results obtained for "perfect hashing" i.e. using uniformly distributed 32 bit random hash extracted independent of the trace. The deviations from the reference line of all the hash functions are similar to the uniform random hash (in some cases even smaller). Hence all the hash functions appear to provide good uniformly distributed output in this case. The differences between the hash functions are much more evident with the real trace. IPSX and MMH conform the worst to the uniform distribution with α values close to the limit of 99.9 % we used. Performance of Bob worsens, but not as severely. CRC32 appears to even "profit" from running on a real trace. However, other tests (not reported here) done modifying some fields of the real trace did not reproduce this "unexpected" gain of CRC32, which therefore probably heavily depends on the nature of the particular trace used. The behavior of Bob, IPSX and MMH was on the contrary stable. On the basis of these hypothesis-based uniformity tests, Bob appears to be the more robust function; CRC32 performances are not stable, while MMH and IPSX have the worst uniformity.

4.2.2 Results for Variability Metric Approach

Figure 4 shows the variability metric (2) for all four candidate hash functions. As expected, if the keys come from a real trace (with less uniformly distributed input), the metric increases (indicating a worse uniformity of the output). But the increase is small and the differences between the 4 tested hashes are limited, indicating that they all do a fairly good job also when the input keys are not uniformly distributed. (In the real trace we had a low variability of the IP addresses, due to the limed number of "speaking hosts" in our lab). IPSX performs slightly worse than the other functions; this was expected because of its simplicity and low number of its operations.

4.3. Hash Functions for Packet Selection – Speed Tests

IPSX (Figure 5) is six times faster than Bob hash, which does mostly Shifts and XOR operations as IPSX, but repeats them more times and also includes additions and subtractions. MMH also does multiplications, and is ten times slower than IPSX. CRC32 is the slowest one because it acts separately on each byte on the input key, while the other functions operate on 4 bytes blocks. These results were obtained doing a profiling of the C software implementation on a PC running Linux. Anyway these results are confirmed by the analysis we did looking at the instructions per byte these functions require to produce the hash result (IPSX: 1.125*n – 1 with n divisible by 8; BOB: 6*n + 35; CRC: 9*n + 3; MMH: 14 * n+15).

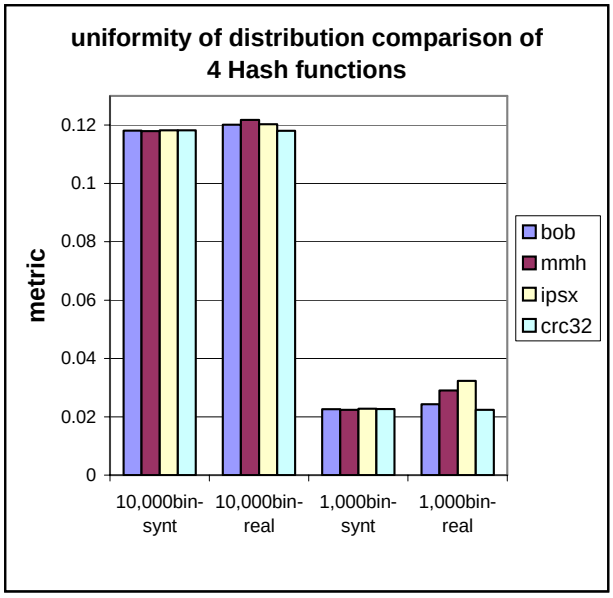

Figure 4. Testing uniformity with metric (2)

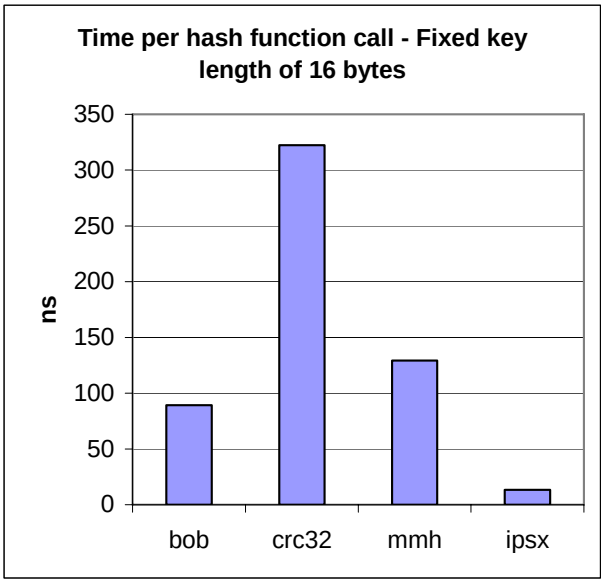

Figure 5. Computation speed

4.4. Hash Functions for Packet Selection – Interim Conclusions

IPSX has definite advantages for computational speed. With a real input trace, the distribution of its hash values is clearly distinguishable from uniform in a significance test (Figure 3), while the variability metric (2) was little different from the other hash functions. This trade-off between computational speed and hash uniformity is expected. It is reasonable that the range of behavior in the significance tests across all the candidate hash functions may be a consequence of the particular trace used. It has lower key variability than might be encountered in a typical WAN trace, since it was collected on a laboratory network that contains few "speaking" hosts. Further experiments in the "intermediate cases" of a real trace possessing more key variability are necessary to draw firmer conclusions. The good performance of IPSX observed in Figure 2 for the synthetic trace are not confirmed by the performance observed in Figure 3. Of the candidate hash functions evaluated, we propose Bob as the best overall for packet selection in high speed links. It has the second ranking in speed and first ranking in uniformity, and has relatively uniform performance over both synthetic and real traffic traces. Although IPSX has the fastest execution, this comes at the cost of reduced uniformity. IPSX may still be useful when computational simplicity is important and with applications that are relatively insensitive to hash non-uniformity. CRC appears to be too slow, and MMH is not a strong candidate having worse speed and uniformity than Bob. It is plausible that results on a trace with more variable keys will not change significantly the relative performance observed in these tests.

4.5. Hash Functions for Digest ID Generation - Collision Tests

As regards as the collision performances, we first noted that with synthetic input keys 16 bytes long all hash functions have a collision probability close to the minimum theoretical one (2.3×10^{-10}, being the hash output on 32 bit). On the contrary, with a real trace, using as input the same 16 bytes we used for the uniformity tests, we verified that collision probabilities of all the functions are above 10^{-8}, which is "high" according to the reasoning done in Section 3 and summarized in Table 1. Moreover, a lot of these collisions are due to identical input keys, so the only way to decrease collision probability is to add bytes to the input key until the number of identical input keys becomes negligible.

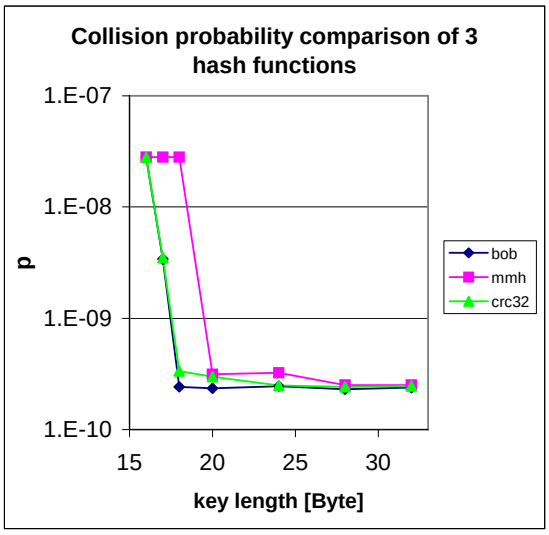

Figure 6. Collision probability tests varying the input key size – Keys extracted from a real trace

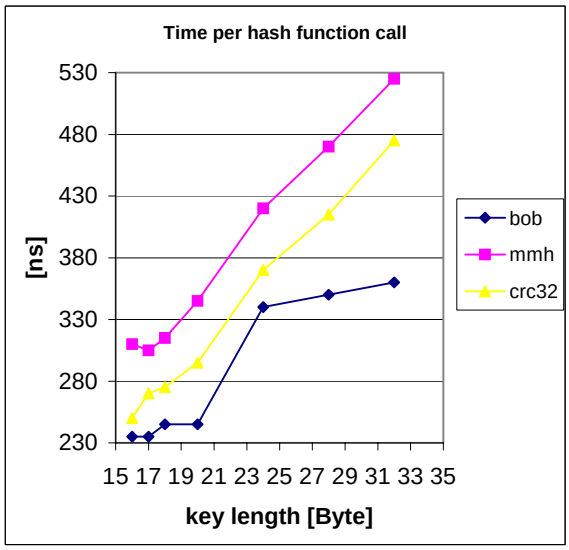

Figure 7. Computation speed for digest ID

This suggests that for the digest ID generation we must use functions that can operate on keys of configurable length, and thus exclude IPSX. Analyzing the collision probability of the other three functions adding additional bytes of the packet payload to the input key, we see (Figure 6) that Bob hash best exploits the variability introduced by these additional bytes. However, for all the functions, beyond a certain input key length the collision probability drops to values close to the theoretical minimum (2.3×10^{-10}) to which they asymptotically tend. Therefore, there is no significant advantage in adding bytes to the input key beyond a certain threshold, also considering that the computation speed of the hash function will increase, as we show in the next section.

4.6. Hash Functions for Digest ID Generation - Speed Tests

A speed comparison of the three hash functions considered for digest ID generation is reported in Figure 7. Note that the times for key lengths of 16 bytes are higher than the ones of Figure 5 because there we tested versions of the hash functions optimized for a fixed length of 16 bytes. Bob is the fastest and the one with the slowest growth in computation time with the input key length, but there is no function really over performing the others (the order of magnitude is the same).

4.7. Hash Functions for Digest ID Generation - Conclusions

On the basis of the tests we concluded that since Bob shows slightly better performances than CRC32, it was wiser to indicate in [9] the former as the preferred hash function for digest ID generation. Even if many software implementations of CRC32 are already available, even the small performance difference

evinced by the tests could justify mandating an additional implementation effort for Bob. This is an implementation choice.

5. CONCLUSIONS AND FUTURE WORK

In this paper we presented a methodology for testing hash function performances when these functions are applied to packet sampling. Using this methodology, we performed a comparison of four hash functions (already selected via a preliminary screening). The results allowed us to identify the hash functions that can best accomplish the tasks of emulating random packet sampling and generating digest IDs for correlating reports of sampled packets at a Data Collector. Concerning statistical properties of hash-based sampling, this work has focused in the uniformity of the generated hash values. In future work, we will investigate how close packet selection comes to the ideal of independent sampling. In our tests we used both synthetically generated keys and keys extracted from packet traces as input for the hash functions. We hope that further tests using the proposed methodology can be carried out with other packet traces in order to confirm the results obtained so far.

REFERENCES

[1] T. Zseby, S. Zander, G. Carle, “Evaluation of Building Blocks for Passive One-way-delay Measurements”, Proceedings of Passive and Active Measurement Workshop (PAM 2001), Amsterdam, The Netherlands, April 23-24, 2001

[2] K.C. Claffy, G.C. Polyzos, H.-W. Braun, ``Application of Sampling Methodologies to Network Traffic Characterization", Computer Communication Review, vol. 23, pp. 194--203, October 1993, also in Proc. ACM SIGCOMM'93, San Francisco, CA, Sept. 13--17, 1993.

[3] N.G. Duffield, “Sampling for Passive Internet Measurement: A Review”, Statistical Science, Vol. 19, No. 3, 472-498, 2004.

[4] N.G. Duffield, A. Gerber, M. Grossglauser,``Trajectory Engine: A Backend for Trajectory Sampling", IEEE Network Operations and Management Symposium 2002, (NOMS 2002) Florence, Italy, April 15-19, 2002.

[5] N. G. Duffield, M. Grossglauser, “Trajectory Sampling for Direct Traffic Observation”, IEEE/ACM Transactions on Networking, 9(3):280-292, June 2001

[6] N.G. Duffield, M.Grossglauser, “Trajectory Sampling with Unreliable Reporting”, IEEE Infocom 2004, Hong Kong, March 7-11, 2004.

[7] B.-Y. Choi, S. Moon, R. Cruz, Z.-L. Zhang, and C. Diot (2003): “Practical Delay Measurement for ISP”, Sprint ATL Research Report Nr. RR03-ATL-051910. Sprint ATL. May 2003.

[8] IETF PSAMP Work. Group, http://www.ietf.org/html.charters/psamp-charter.html

[9] T.Zseby, et al., “Sampling and Filtering Techniques for IP Packet Selection” http://www.ietf.org/internet-drafts/draft-ietf-psamp-sample-tech-06.txt, Feb. 2005. – Work in progress

[10] N.G. Duffield (Ed.) “A Framework for Packet Selection and Reporting” draft-psamp-framework-08.txt, Sept. 2004, Work in progress

[11] S.Niccolini, M.Molina, S.Tartarelli, F.Raspall “Design and implementation of a One Way Delay passive measurement system” – IEEE Network Operations and Management Symposium 2004, Korea, Apr. 2004

[12] R. Braden, D. Borman, C. Partridge, “Computing the Internet Checksum”, RFC 1071, Sep. 1988 (updated by RFCs 1141 and 1624)

[13] S. Halevi, H. Krawczyk, “MMH: Software Message Authentication in the Gbit/second Rates” 4th Workshop on Fast Software Encryption, pages 172-189. Springer, LNCS vol. 1267, 1997

[14] Bob Jenkins’ hash function web page, paper published in Dr Dobb’s journal, http://burtleburtle.net/bob/hash/doobs.html

[15] On time synchronization and hashing for passive One-Way Delay measurements. NEC Internal, June 2003.

[16] A. Feldmann, et al., “Deriving Traffic Demands for Operational IP Networks: Methodology and Experience”, IEEE/ACM Transactions on Networking, June 2001.

[17] Synergy Systems LLC Home page, Motorola M12+Timing Starter kit, http://www.synergy-gps.com/

[18] D. Pezaros, D. Hutchison, F. Garcia, R. Gardner, J. Sventek, In-line Service Measurements: An IPv6-based Framework for Traffic Evaluation and Network Operations, IEEE NOMS 2004, Seoul Korea.