Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2006-033 April 29, 2006

# Supplement to "Distributed Quota Enforcement for Spam Control"

Michael Walfish, J.D. Zamfirescu, Hari Balakrishnan, David Karger, and Scott Shenker

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

# Supplement to "Distributed Quota Enforcement for Spam Control"

Michael Walfish[*], J.D. Zamfirescu[*], Hari Balakrishnan[*], David Karger[*], and Scott Shenker[†]

## Abstract

This report is a supplement to our paper "Distributed Quota Enforcement for Spam Control" [1]. We assume here that the reader has read the main paper. In this report, we first analyze the enforcer nodes' key-value maps and then analyze two of the experiments from the main paper.

## 1 Analysis of Enforcer Nodes' Key-Value Maps

The key-value map used by an enforcer node is described in §4.2 of [1]. The map consists of an on-disk log and an in-memory index. The index is itself two pieces: a modified open addressing hash table and an overflow table. Recall that a new key is usually inserted in the hash table but that a collision (which happens when the probe sequence induced by the key hits an entry that has the same 8-bit checksum as the key) sends the key into the overflow table.

In this section, we analyze the the hash table and the overflow table. We use the standard assumption that hash functions map each key to an random output, and in particular that the probe sequence for each key is an independent random sequence. We also assume that the checksum is an independent random value. Let $\alpha < 1$ be the load factor (*i.e.*, ratio of non-empty entries to total entries) of the hash table. Let $N$ be the number of keys that a node will store. We pessimistically assume that all $N$ keys are already in the index.

We first calculate the probability that a key will be inserted in the overflow table. Consider a key, $k$, that the node is about to insert in the index. Each position in the probe sequence is empty with probability $1-\alpha$. If the entry is not empty, then it has a matching checksum with probability $1/c$, where $c$ is the number of distinct checksum values (256 in our case). So a probe has one of three possible outcomes: empty (with probability $1-\alpha$), matching checksum (with probability $\alpha/c$) and non-matching checksum (with probability $\alpha(1-1/c)$). The node stops probing when one of the first two cases applies. The probability of the second case (matching checksum, which forces $k$ into the overflow table) conditioned on the event that the node stopped probing is equal to the probability of the second case divided by the probability of the first two cases, namely

$$\begin{aligned} \frac{\alpha/c}{1-\alpha+\alpha/c} &= \frac{1}{c(1/\alpha-1)+1} \\ &\leq \frac{1}{(1-\alpha)c}. \end{aligned}$$

Thus, under our pessimistic assumption, $k$ has worst-case probability $\frac{1}{(1-\alpha)c}$ of winding up in the overflow table. Then, if the node stores $N$ keys, the expected number of keys in the overflow table is at most $\frac{N}{(1-\alpha)c}$.

[*]MIT Computer Science and AI Lab. Email: {mwalfish,zamfi,hari,karger}@csail.mit.edu.
[†]UC Berkeley and ICSI. Email: shenker@icsi.berkeley.edu.

Each entry in the overflow table takes up 24 bytes (20 bytes for the key and 4 bytes for the disk offset). Since the hash table has size $N/\alpha$ and since each entry is exactly 4 bytes, the total size of the structure is

$$N\left(\frac{4}{\alpha} + \frac{24}{(1-\alpha)c}\right).$$

Taking $c = 256$, the expression above is minimized for $\alpha^* = .847$.[1] Plugging in $\alpha^*$, the above expression becomes $5.34N$. Thus, each key costs roughly 5.34 bytes in expectation or roughly 1.3 four-byte words, and we get the following claims:

**Claim 1** *The value of $\alpha$ that minimizes the total RAM requirement of the hash table and overflow table is* 0.847.[2]

**Claim 2** *The RAM cost of the index is* $1.3N$ *four-byte words.*

To calculate the expected number of lookups per key, we must determine how many items in the probe sequence the node must inspect before finding an empty one. Since each entry is empty with probability $1 - \alpha$, a node expects to inspect $1/(1-\alpha)$ entries before finding the desired key or discovering it is absent. Thus, for $\alpha = 0.847$, we get the following claim:

**Claim 3** *To find the checksum for a given key, k, (or to discover that k is not stored in the hash table), a node must investigate an average of 6.5 positions in the probe sequence induced by k.*

## 2 Exact Expectation Calculation in "Crashed" Experiment

In this section, we derive an exact expression for expected stamp use in the "crashed" experiment from §6.2 of [1]. (The expression is stated in footnote 7.) Recall from that section that $n$ is the number of nodes in the system, $p$ is the probability a machine is "bad" (*i.e.*, does not respond to queries), $m = n(1-p)$ is the number of "up" or "good" machines, stamps are queried 32 times, and $r$, the replication factor, is 3.

**Claim 4** *The expected number of uses per stamp in the "crashed" experiment from §6.2 is:*

$$\begin{aligned}
& (1-p)^3(1) + 3p^2(1-p)\alpha + 3p(1-p)^2\beta + p^3 m\left(1 - \left(\frac{m-1}{m}\right)^{32}\right), \quad \textit{for} \\
\alpha &= \sum_{i=1}^{m} i\left(\frac{2}{3}\right)^{i-1}\frac{1}{m}\left(1 + \frac{m-i}{3}\right), \\
\beta &= \sum_{i=1}^{m-1} i\left(\frac{1}{3}\right)^{i-1}\frac{m-i}{m(m-1)}\left(2 + \frac{2}{3}(m-(i+1))\right)
\end{aligned}$$

**Proof:**

To prove this claim, we consider 4 cases: 0 of a stamp's 3 assigned nodes are good; 1 is good; 2 are good; all 3 are good.

[1] The version of [1] that appears in the USENIX NSDI conference proceedings erroneously reports $\alpha^* = .87$. The version on our Web site (`http://nms.csail.mit.edu/dqe`) is correct, and we have updated the online copy with USENIX.

[2] If the overflow table is implemented as a data structure other than a flat table, then it would require additional bytes of overhead. This overhead would affect the RAM cost and the optimal $\alpha$ but the effect is not major.

Let $U(s)$ be the number of times a stamp $s$ is used. We calculate the expected value of $U(s)$ in each of the four cases. The first case is trivial: if all of $s$'s assigned nodes are good (which occurs with probability $(1-p)^3$), the stamp will be used exactly once.

Next, to determine $\mathbb{E}[U]$ for stamp with no good assigned nodes (probability $p^3$), we recall the facts of the experiment: stamps are queried 32 times at random portals, and once a stamp has been SET at a portal, no more reuses of the stamp will occur *at that portal*. Thus, the expected number of times that $s$ will be used, if *none* of its assigned nodes is good, is the expected number of distinct bins (out of $m$) that 32 random balls will cover. Since the probability a bin isn't covered is $\left(\frac{m-1}{m}\right)^{32}$, the expected value of $U(s)$ in this case is:

$$m\left(1-\left(\frac{m-1}{m}\right)^{32}\right).$$

We now compute the expected number of stamp uses for stamps with one or two good assigned nodes. In either case:

$$\mathbb{E}[U] \quad = \quad 1\cdot\Pr(\textit{exactly 1 use}) + 2\cdot\Pr(\textit{exactly 2 uses}) + \cdots$$

For stamps with one good assigned node (probability $(1-p)p^2$) there are two ways for the stamp to be used exactly once: either, with probability $\frac{1}{m}$, the stamp is TEST and then SET at the one good assigned node, or, with probability $\left(\frac{m-1}{m}\right)\frac{1}{3}$, the PUT generated by the SET is sent to the good assigned node. (The latter probability is the product of the probabilities that the TEST and SET are sent to a node *other than* the good assigned node and that the resulting PUT is sent to the good assigned node.) Thus, $\Pr(\textit{exactly 1 use}) = \frac{1}{m} + \left(\frac{m-1}{m}\right)\frac{1}{3}$.

If the stamp is used exactly twice, then the stamp was not stored at its good assigned node on first use; this occurs with probability $\left(\frac{m-1}{m}\right)\frac{2}{3}$. To calculate the probability that the second use is the last use, we apply the same logic as in the *exactly 1 use* case. Either, with probability $\frac{1}{m-1}$, the stamp is TEST and SET at the good assigned node ($m-1$ because there has already been one use, so one of the $m$ nodes already stores the stamp, and thus a TEST at that node would not have resulted in this second use), or, with probability $\left(\frac{m-2}{m-1}\right)\frac{1}{3}$, the PUT generated by the SET is sent to the good assigned node. Thus, $\Pr(\textit{exactly 2 uses}) = \left(\frac{m-1}{m}\right)\frac{2}{3}\left[\frac{1}{m-1} + \left(\frac{m-2}{m-1}\right)\frac{1}{3}\right]$.

By the same logic, a third use only happens if the first and second uses do not store the stamp on the good node, and the third use is the last use if it results in the stamp being stored on its good assigned node: $\Pr(\textit{exactly 3 uses}) = \left(\frac{m-1}{m}\right)\frac{2}{3}\left(\frac{m-2}{m-1}\right)\frac{2}{3}\left[\frac{1}{m-2} + \left(\frac{m-3}{m-2}\right)\frac{1}{3}\right]$. A pattern emerges; cancellation of terms yields an expression for the general case: $\Pr(\textit{exactly i uses}) = \left(\frac{2}{3}\right)^{i-1}\frac{1}{m}\left(1+\frac{m-i}{3}\right)$.

Thus we have an expression for the expected number of uses for stamps with one good node:

$$\mathbb{E}[U] = \sum_{i=1}^{m} i\left(\frac{2}{3}\right)^{i-1}\frac{1}{m}\left(1+\frac{m-i}{3}\right) \tag{1}$$

A similar argument applies to stamps with two good nodes (probability $(1-p)^2p$), except we begin with $\Pr(\textit{exactly 1 use}) = \frac{2}{m} + \left(\frac{m-2}{m}\right)\frac{2}{3}$. The $\frac{2}{m}$ term replaces $\frac{1}{m}$ because a TEST and SET to either of the (now two) good assigned nodes will result in exactly 1 use, and $\frac{2}{3}$ replaces $\frac{1}{3}$ because the SET's PUT now has a $\frac{2}{3}$ chance of reaching a good assigned node.

To get $\Pr(\textit{exactly 2 uses})$, we follow similar logic as before. The first use is not the last with probability $\left(\frac{m-2}{m}\right)\frac{1}{3}$, as the stamp is SET to a non-assigned node with probability $\frac{m-2}{m}$ and PUT to a bad node with probability $\frac{1}{3}$. Then, the second use is the last with probability $\frac{2}{m-1} + \left(\frac{m-3}{m-1}\right)\frac{2}{3}$, and $\Pr(\textit{exactly 2 uses}) = \left(\frac{m-2}{m}\right)\frac{1}{3}\left[\frac{2}{m-1} + \left(\frac{m-3}{m-1}\right)\frac{2}{3}\right]$.

Continuing, $\Pr(\textit{exactly 3 uses}) = \left(\frac{m-2}{m}\right)\frac{1}{3}\left(\frac{m-3}{m-1}\right)\frac{1}{3}\left[\frac{2}{m-2} + \left(\frac{m-4}{m-2}\right)\frac{2}{3}\right]$. Again, a pattern emerges, and cancellation yields $\Pr(\textit{exactly i uses}) = \left(\frac{1}{3}\right)^{i-1}\frac{m-i}{m(m-i)}\left(2+\frac{2}{3}(m-(i+1))\right)$.

Thus, the expected number of uses for a stamp with two good nodes is:

$$\mathbb{E}[U] = \sum_{i=1}^{m-1} i\left(\frac{1}{3}\right)^{i-1} \frac{m-i}{m(m-1)}\left(2 + \frac{2}{3}(m-(i+1))\right) \tag{2}$$

Note that for equations 1 and 2, the summation begins with the first use ($i = 1$) and ends with the stamp being on as many nodes as possible ($i = m$ or $i = m - 1$).

Letting $\alpha = \sum_{i=1}^{m} i\left(\frac{2}{3}\right)^{i-1} \frac{1}{m}\left(1 + \frac{m-i}{3}\right)$ (from equation 1) and $\beta = \sum_{i=1}^{m-1} i\left(\frac{1}{3}\right)^{i-1} \frac{m-i}{m(m-1)}\left(2 + \frac{2}{3}(m-(i+1))\right)$ (from equation 2), we get the claim, which justifies the expression from footnote 7 of [1].

■

# 3 Relating Microbenchmarks to System Performance

In this section we calculate how many RPCs are induced by a TEST in expectation.

**Claim 5** *The number of RPCs per* TEST *in the 32-node enforcer experiments described in §6.4, in which 50% of* TEST *requests are "fresh", is 9.95, on average.*[3]

**Proof:**

Recall from §6 that the 32-node enforcer is configured with replication factor $r = 5$. On receiving a fresh TEST, the portal must contact all 5 assigned nodes for the stamp. With probability 5/32, the portal is an assigned node for the stamp, and one of the GETs will be local. Thus, we expect a fresh TEST to generate $\frac{5}{32} \cdot 4 + \frac{27}{32} \cdot 5 = 4.84$ GET requests and GET responses. (Note that a request and a response both cause the CPU to do roughly the same amount of work, and thus an RPC response counts as an RPC in our calculations.) A fresh TEST will also be followed by a SET that will in turn cause both a PUT and a PUT response with probability 31/32 = 0.97. (With probability 1/32, the portal is one of the assigned nodes and chooses itself as the node to PUT to, generating no remote PUT.)

A reused TEST generates no subsequent SET, PUT request, or PUT response. In addition, for reused TESTs, the number of induced GETs is less than in the fresh TEST case: as soon as a portal receives a "found" response, it will not issue any more GETs. The exact expectation of the number of GETs caused by a reused TEST, 2.64, is derived below.

| RPC type | Fresh | Reused | Average |
|---|---|---|---|
| TEST | 1.0 | 1.0 | 1.0 |
| GET | 4.84 | 2.64 | 3.74 |
| GET resp. | 4.84 | 2.64 | 3.74 |
| SET | 1.0 | 0 | 0.5 |
| PUT | 0.97 | 0 | 0.485 |
| PUT resp. | 0.97 | 0 | 0.485 |
| Total RPCs/TEST | | | 9.95 |

Table 1: RPCs generated by fresh and reused TESTs.

The types and quantities of RPCs generated are summarized in Table 1; the average number of RPCs generated per TEST assumes that 50% of TESTs are fresh and 50% are reused, as in the experiment from

[3]The version of [1] that appears in the USENIX NSDI conference proceedings erroneously reports 10.11 instead of 9.95. As with the correction mentioned above, the version on our Web site is correct, and we have updated the online copy with USENIX.

§6.4. Thus, the expected number of RPCs generated by a single TEST is:

$$1.0 + \frac{1}{2}\left[\left(\frac{5}{32}\cdot 4 + \frac{27}{32}\cdot 5\right) + 2.64 + \left(\frac{5}{32}\cdot 4 + \frac{27}{32}\cdot 5\right) + 2.64 + 1 + \frac{31}{32} + \frac{31}{32}\right] = 9.95$$

■

**Claim 6** *A reused* TEST *generates 2.64* GET*s in expectation.*

**Proof:**

The number of GETs generated by a TEST for a reused stamp depends on the circumstances of the stamp's original SET: did the SET occur at an assigned node, and if so, did it induce a remote PUT? Note that, for any stamp, 27 of the 32 enforcer nodes will not be assigned nodes. Thus, with probability $\frac{27}{32}$, a SET will be to a non-assigned node, and the stamp will be stored at both an assigned node and a non-assigned node (event $A_1$). If the SET occurs at an assigned node (with probability $\frac{5}{32}$), then $\frac{1}{5}$ of the time the node will choose itself as the recipient of the PUT (event $A_2$, with overall probability $\frac{1}{5}\cdot\frac{5}{32} = \frac{1}{32}$), and the stamp will only be stored at that single, assigned node; $\frac{4}{5}$ of the time, the node will choose another assigned node (event $A_3$, with overall probability $\frac{4}{5}\cdot\frac{5}{32} = \frac{4}{32}$), and the stamp will be stored at two assigned nodes. We summarize the three possible circumstances in Table 2. Note that the events $A_i$ partition their sample space.

| *Name* | Pr($A_i$) | stamp originally SET at ... |
|---|---|---|
| $A_1$ | 27/32 | ... a non-assigned node |
| $A_2$ | 1/32 | ... an assigned node, no further PUTs |
| $A_3$ | 4/32 | ... an assigned node, 1 additional PUT |

Table 2: Possible SET circumstances.

The number of GETs caused by a TEST for a reused stamp also depends on the circumstances of the TEST: is the queried node storing the stamp, and if not, is the node one of the stamp's assigned nodes? There are again three possible circumstances: the TEST is sent to some node storing the stamp (event $B_1$); the TEST is sent to an *assigned* node not storing the stamp (event $B_2$); the TEST is sent to a *non-assigned* node not storing the stamp (event $B_3$). These events are summarized in Table 3; they partition their sample space as well.

| *Name* | stamp queried (TESTed) at ... |
|---|---|
| $B_1$ | ... a node storing the stamp |
| $B_2$ | ... an assigned node not storing the stamp |
| $B_3$ | ... a non-assigned node not storing the stamp |

Table 3: Possible reused TEST circumstances.

Now, let $C(A_i, B_j)$ count the number of GET RPCs that occur when events $A_i$ and $B_j$ are true. Values of $C(A_i, B_j)$ are easy to determine. First consider event $B_1$: the TEST is sent to a node already storing the stamp. In this case, there will be no remote GETs regardless of the original SET's results. Next, consider event $B_2$: the TEST is sent to an assigned node not storing the stamp; now, events $A_1$ and $A_2$ both cause a single assigned node to store the stamp, and thus, in either case, we expect the portal to send 2 (of $r - 1 = 4$ possible) GETs. However, event $A_3$ causes the stamp to be stored on two assigned nodes, and we expect the portal to send $\left(\frac{1}{2}\right)\cdot 1 + \left(1-\frac{1}{2}\right)\left(\frac{2}{3}\right)\cdot 2 + \left(1-\frac{1}{2}\right)\left(1-\frac{2}{3}\right)(1)\cdot 3 = 1 + \frac{2}{3}$ GETs. Finally, consider event $B_3$: the TEST is set to a non-assigned node not storing the stamp. If the stamp is stored on a single assigned node (events $A_1$, $A_2$), we expect the portal to send 3 (of 5 possible) GETs; if the stamp is stored on two assigned nodes ($A_3$), we

| $C(A_i, B_j)$ | $A_1$ | $A_2$ | $A_3$ |
|---|---|---|---|
| $B_1$ | 0 | 0 | 0 |
| $B_2$ | 2.5 | 2.5 | (1+2/3) |
| $B_3$ | 3 | 3 | 2 |

Table 4: Values of $C(A_i, B_j)$, the expected number of RPCs generated by a TEST when $A_i$ and $B_j$ are true.

expect the portal to send $\left(\frac{2}{5}\right) \cdot 1 + \left(1 - \frac{2}{5}\right)\left(\frac{2}{4}\right) \cdot 2 + \left(1 - \frac{2}{5}\right)\left(1 - \frac{2}{4}\right)\left(\frac{2}{3}\right) \cdot 3 + \left(1 - \frac{2}{5}\right)\left(1 - \frac{2}{4}\right)\left(1 - \frac{2}{3}\right)(1) \cdot 4 = 2$ GETs. We summarize the values of $C(A_i, B_j)$ in in Table 4.

Now we can construct an expression for the expected number of RPCs generated by a reused TEST, which we call $C$:

$$C = \sum_{j=1}^{3} \sum_{i=1}^{3} C(A_i, B_j) \cdot \Pr(A_i \wedge B_j). \tag{3}$$

To calculate this expression, we use $\Pr(A_i \wedge B_j) = \Pr(B_j \mid A_i) \cdot \Pr(A_i)$. We know the value of each $\Pr(A_i)$, so we are left with finding each $\Pr(B_j \mid A_i)$. We begin by considering the stamps originally SET at a non-assigned node (event $A_1$), which are now stored at one assigned node and one non-assigned node. Given event $A_1$, there are 2 nodes storing the stamp, 4 assigned nodes not storing the stamp, and 26 non-assigned nodes not storing the stamp. The probability of sending a TEST to nodes in these three classes, which correspond to events $B_1$, $B_2$, and $B_3$, respectively, are simply 2/32, 4/32, and 26/32. The same method can be used to find the conditional probabilities given $A_2$ and $A_3$; we present these values in Table 5.

| $\Pr(B_j \mid A_i)$ | $A_1$ | $A_2$ | $A_3$ |
|---|---|---|---|
| $B_1$ | 2/32 | 1/32 | 2/32 |
| $B_2$ | 4/32 | 4/32 | 3/32 |
| $B_3$ | 26/32 | 27/32 | 27/32 |

Table 5: Conditional probabilities $\Pr(B_j \mid A_i)$.

Combining the values of $C(A_i, B_j)$ with the joint probabilities, we compute, from equation (3), $C = 2.64$.

■

# References

[1] M. Walfish, J. D. Zamfirescu, H. Balakrishnan, D. Karger, and S. Shenker. Distributed Quota Enforcement for Spam Control. In *NSDI*, May 2006.