Technical Disclosure Commons

Defensive Publications Series

April 2020

COMPUTE N-WAY DE-DUPLICATED REACH USING PRIVACY SAFE VECTOR OF COUNTS

Sheng Ma

Laura Book

Xichen Huang

Joey Knightbrook

Scott Schneider

See next page for additional authors

Inventor(s)
Sheng Ma, Laura Book, Xichen Huang, Joey Knightbrook, Scott Schneider, Jiayu Peng, and Michael Daub

COMPUTE N-WAY DE-DUPLICATED REACH USING PRIVACY SAFE VECTOR OF COUNTS

SUMMARY

This disclosure relates to the computation of the de-duplicated reach of user identifiers across multiple parties in a privacy safe way. Other methods, such as HyperLogLog, can compute the de-duplicated union of a multiset, but also expose information about the members of those multisets. This present issues to privacy when those other methods are used to compute the reach of user identifiers. The solution described herein allows for the independent computation of a vector of counts by each publisher, which can represent the aggregate set of user identifiers of the publisher without exposing any information about the user identifiers themselves. Using statistical analysis, this solution can compute the de-duplicated union of sets of user identifiers across two or more publishers using these vectors of counts without exposing information about the user identifiers to any outside party.

DESCRIPTION

This disclosure presents a method for calculating the de-duplicated reach of a content campaign that provides increased user privacy protection over existing methods. This method can support third-party measurement providers by enabling reach measurement accuracy, reasonable compute time and space consumption, and the ability to de-duplicate between multiple publishers without compromising the security of the system or the privacy of users.

It has been shown that cardinality estimation methods that are arbitrarily aggregatable, such as HyperLogLog (HLL), are incompatible with privacy guarantees when sharing their estimation sketches directly. Other solutions for privacy-preserving cardinality estimation, such as Pan-Private Bloom filters or privacy-preserving PCSA, do not allow for simultaneously high accuracy and privacy. The Vector of Counts (VoC) method can achieve high accuracy and privacy when de-duplicating between two or more publishers. In this disclosure, we present the basic methodology to compute the de-duplicated union of sets user identifiers across two publishers, and an extension to this implementation that can calculate the de-duplicated union of sets of user identifiers across more than two publishers.

Sketch Construction

Given a set of N user IDs, we generate a sketch, an array of size $k=2^j$ for integer j, that compactly represents that set. The set is first de-duplicated; each distinct user is then hashed using an agreed per-sketch secret salt. The last j bits of the hashed value serve as the index of the bucket in the sketch to be incremented for this element (note that using the leading bits of the hashed value works equally well for reasonable choices of hash function). We then add Laplacian noise of scale $b=1/\varepsilon$ to each bucket in the sketch. Each register of the resulting vector can thus contain the count of distinct users mapping to that bucket plus noise. Figure 1 below details the construction of the vector of counts.

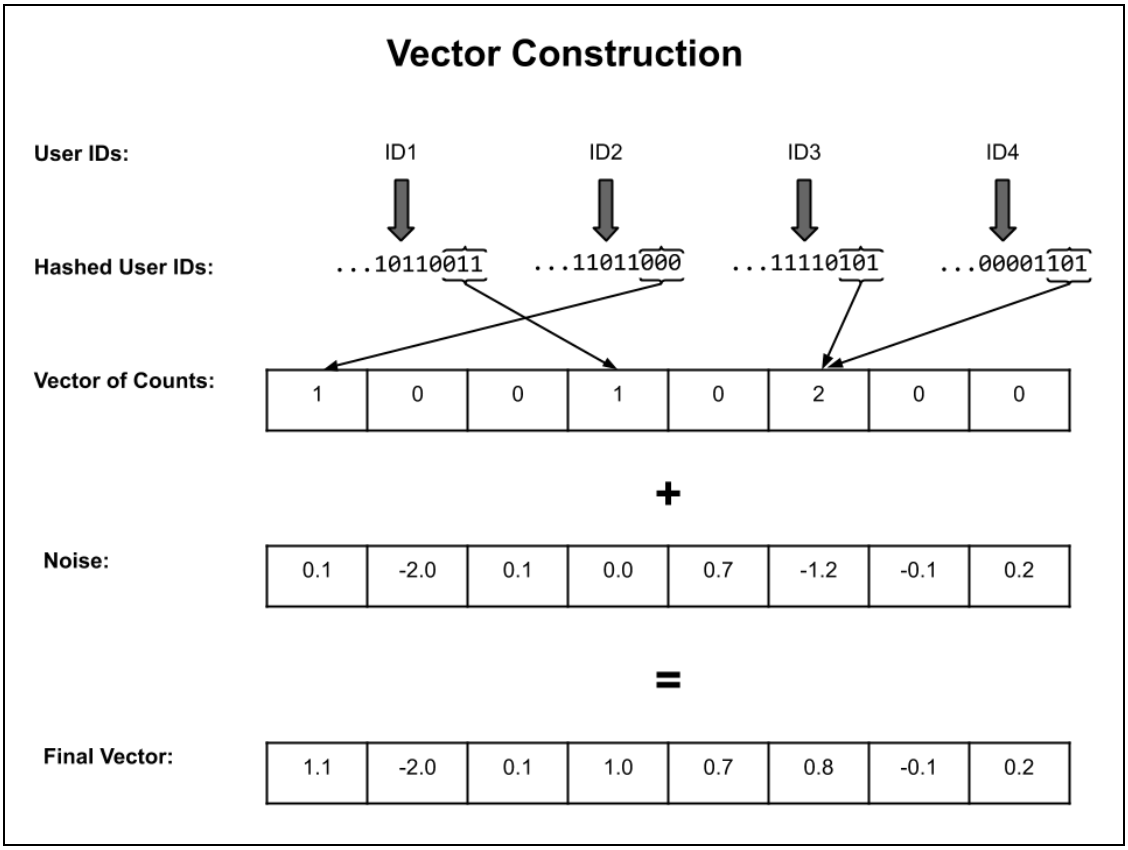

Figure 1 – Construction of a Vector of Counts

Exemplary pseudo-code to generate such a vector is included below.

```
def ComputeVectorOfCounts(k, b, user_set):
  """
  Args:
    k: Size of the vector to be returned.
    b: Scale factor of the Laplacian noise.
```

```
    user_set: Deduplicated set of user IDs.

  Returns:
    The vector of counts of size k for the given user set, with
Laplacian
    noise of scale b added.
  """
  hashed_user_set = get_hashed_user_set(user_set)
  user_buckets = [get_last_k_digits(id, k) for id in hashed_user_set]

  voc = []
  for i in range(k):
    voc.append(user_buckets.count(i) + generate_laplace_noise(b))
  return voc
```

Similar sets of users can have similar vectors, but a large subset of user IDs get mixed together because they have the same bucket index. That mixing, as well as the per-bucket noise, ensures privacy at the expense of reduced accuracy when calculating the intersection of these vectors for reach estimation.

Reach Estimation

The de-duplicated cardinality of the union of two user sets can be estimated from vectors of counts *c1* and *c2* constructed as described above for each of the sets. Note that here expectation values are taken over varying both the noise and the choice of hash function. The total number of users represented in each sketch can be provided with the sketch; we use this to first subtract off the expectation value of each register. Denoting element j of vector c_i as $c_i(j)$, and the total count of vector c_i as N_i,

$$v_i = c_i - \frac{N_i}{k}$$

Each set can consist of a subset of users that can be shared between the two sets and a unique set of users not contained in the other set. Each vector is therefore the sum of vectors for the shared part z, the unique part u_i, and noise e_i:

$$v_i = z + u_i + e_i \quad \text{..... (1)}$$

Taking the expectation value of the dot product of the two mean-subtracted vectors:

$$\begin{aligned} E(v_1 \cdot v_2) &= E[(z + u_1 + e_1) \cdot (z + u_2 + e_2)\,] \\ &= E(z \cdot z) + E(z \cdot u_1) + E(z \cdot u_2) + E(u_1 \cdot u_2) + E(z \cdot e_1) + E(z \cdot e_1) + E(u_2 \cdot e_1) + E(u_1 \cdot e_2) + E(e_1 \cdot e_2) \\ &\simeq E(z \cdot z) \end{aligned}$$

Here, the noise terms are independent from all of the other terms and are drawn from zero-centered distributions, so the expectation value of their product with any other term is zero. Similarly, the mean-subtracted vectors of counts for disjoint user sets are independent from each other, so the expectation values of their dot products are also zero. Given a uniform sampling of the user ID space, each bucket is chosen with probability $1/k$ for each of the N_i items in sketch i. As a result, the counts in each sketch register are approximately a binomial distribution with probability $1/k$ and number of trials N_i. For large counts, this is well approximated by a Gaussian distribution with variance.

$$Var\left[v_i(j)\right] = N_i\,(k-1)/k^2 \quad \text{..... (2)}$$

The mean of the distribution is zero since we have subtracted it off. The expectation value of the dot product norm of a mean subtracted vector of counts is then:

$$\begin{aligned} E(z \cdot z) &= \sum_{j=1}^{k} E\left[z(j)^2\right] \\ &= \sum_{j=1}^{k} Var\,(z(j)) \\ &= \frac{N_{12}\,(k-1)}{k} \\ &\approx N_{12} \end{aligned}$$

where N_{12} is the number of items in the intersection of the two sets. In the cases where the size of the vectors k is large, $(k\text{-}1)/k \approx 1$. Therefore, we can estimate the size of the intersection between two sets using the dot product of their mean-subtracted vectors of counts:

$$\widehat{N_{12}} = v_1 \cdot v_2 \quad \text{..... (3)}$$

Pseudo-code that can compute the intersection could look like the following:

```
def ComputeVocIntersection(voc1, voc2, n1, n2, k):
  """
  Args:
    voc1, voc2: Vectors of counts for sets 1 and 2.
    n1, n2: Cardinalities of sets 1 and 2.
    k: Size of the vectors of counts.

  Returns:
    The cardinality of the intersection of the two sets.
  """
  assert len(voc1) == len(voc2) == k
  return sum((voc1[i] - n1/k) * (voc2[i] - n2/k) for i in range(k))
```

The variance of this estimator is given by:

$$Var(\widehat{N_{12}}) = \frac{N_1 N_2 + N_{12}^2}{k} + \frac{2(N_1+N_2)}{\varepsilon^2} + \frac{4k}{\varepsilon^4}$$

The de-duplicated cardinality of the union of the two sets can then be estimated using the intersection size and the inclusion-exclusion principle. In the above algorithm for computing set intersection cardinality, we assumed that both sketches were of the same size. However, the variance of our intersection size estimate is related to the cardinalities of the two sets (equation (4)). Variance increases for sketches with both too large and too small sizes due to increasing contributions from Laplacian noise and register overfilling, respectively. The publishers can choose the size k of each sketch to minimize intersection cardinality estimate variance. At least one optimal sketch size is given by the following equation.

$$k_{optimal} = \frac{\varepsilon^2}{2}\sqrt{N_1 N_2 + N_{12}^2}$$

However, publishers can choose their sketch sizes independently given their own reach. Therefore, sketches of different sizes can be modified before being compared to accommodate different size choices from different publishers. A larger sketch of size k_2 can be down-sampled to the size of a smaller sketch k_1 by summing the values in registers congruent mod k_1 (see the diagram below). To accommodate this, returned sketch sizes should be powers of 2. Note that downsampling a noised vector of counts can effectively multiply the variance of the noise by

k_2/k_1, since each register of the down-sampled vector can contain the sum of k_2/k_1 counts with noise already applied.

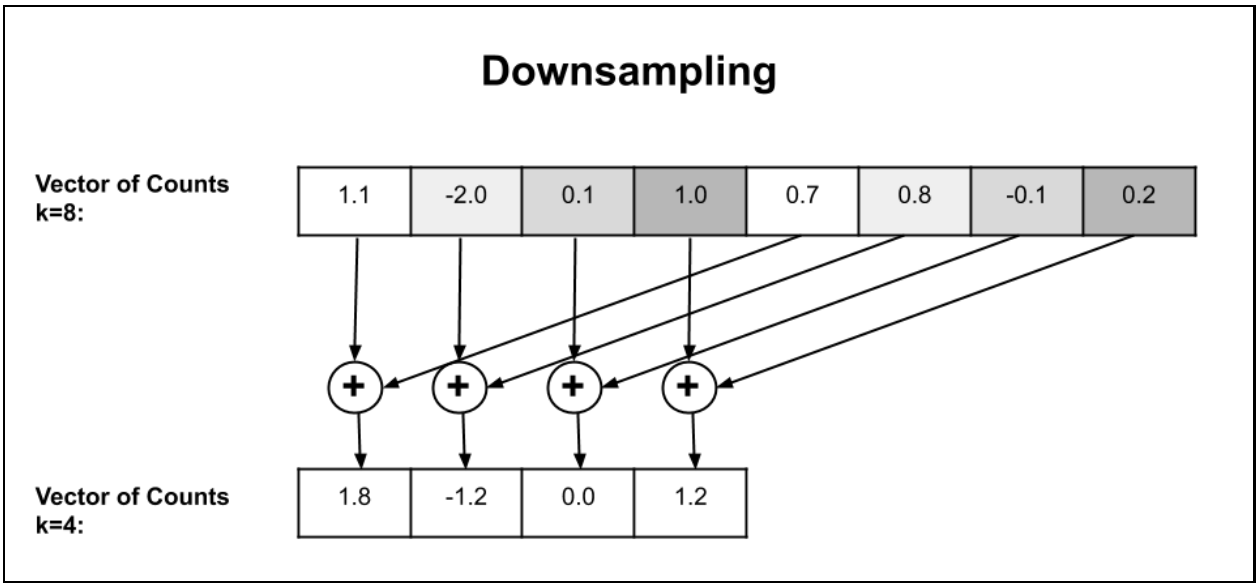

Figure 2 – Downsampling Vector of Counts

Making the assumption that the cardinalities of the two sets are both *N*, at least one optimal vector size is given by the following equation.

$$k_{optimal} = \frac{N}{2}\epsilon^2$$

Sketches sized to a power of 2 near this value are shown to get good accuracy. A few other considerations affect the choice of vector size. Smaller vectors can be faster to compute and transmit (and slightly more privacy safe), so we may use a size 2-4x below the optimal value, so long as it does not increase errors too much. For the same reasons, we may also set a maximum vector size. Finally, the system may not return vectors with too few users, as these would have very low accuracy and limited privacy safety.

Multiple Publisher Case

This section describes various methods for computing the de-duplicated reach using the Vector of Counts approach across multiple publishes (e.g., more than two).

A. Method 1: Inclusion-Exclusion Method

The inclusion-exclusion method can determine the union size from the size of each of the intersections of the sets. An example of such computation is given in the following equation.

$$|A \cup B \cup C| = |A|+|B|+|C|-|A \cap B|-|A \cap C|-|B \cap C|+|A \cap B \cap C|$$

With two-sets, the formula is this:

$$|A \cup B| = |A|+|B|-|A \cap B|$$

And for 3 sets you have:

$$|A \cup B \cup C| = |A|+|B|+|C|-|A \cap B|-|A \cap C|-|B \cap C|+|A \cap B \cap C|$$

For 4 sets, it starts to get hairy:

$$|A \cup B \cup C \cup D| = |A|+|B|+|C|+|D|-|A \cap B|-|A \cap C|-|A \cap D|-|B \cap C|-|B \cap D|-$$
$$|C \cap D|+|A \cap B \cap C|+|A \cap B \cap D|+|A \cap C \cap D|+|B \cap C \cap D|-|A \cap B \cap C \cap D|$$

The general pattern, which always holds, is to add the odd-ordered intersections, and subtract the even-ordered intersections. With Vector of Counts, if the vectors are large enough, the system can calculate the three-way intersections from the raw vectors and put those into the inclusion-exclusion formula to get a better approximation of the union size. However, a better approximation to the inclusion-exclusion is as follows.

Instead of multiplying the even intersections by -1 and the off intersections by +1, you calculate floating-point coefficients as a function of the total number of sets you have, and the highest-order-available intersections size. As an example, the formula emits this for 4 sets and only pairwise intersections:

$$|A \cup B \cup C \cup D| =$$
$$0.71428571[|A|+|B|+|C|+|D|]$$
$$-0.28571429*[|A \cap B|+|A \cap C|+|A \cap D|+|B \cap C|+|B \cap D|+|C \cap D|]$$

The generic derived coefficients then give way to an error bound provided by the following equations.

1. *For* $k \geq \Omega(\sqrt{n})$

$$\frac{|\bigcup_{i=1}^{n} A_i|}{|\bigcup_{i=1}^{n} B_i|} = 1 + O(e^{-\frac{2k}{\sqrt{n}}})$$

2. *For* $k \leq O(\sqrt{n})$

$$\frac{|\bigcup_{i=1}^{n} A_i|}{|\bigcup_{i=1}^{n} B_i|} = O(\frac{n}{k^2})$$

Example code to implement the inclusion-exclusion principle approximations is provided below.

```
def voc_inclusion_exclusion_clever_truncation(voc_list, max_order):
  assert max_order <= MAX_SUPPORTED_VOC_OVERLAP
  num_pubs = len(voc_list)

  if num_pubs == 1:
    return voc_overlap(voc_list)

  max_considered_order = min(max_order, num_pubs)
  relevant_alpha_vector = alpha_vector(max_considered_order, num_pubs)

  union_estimate = 0
  for current_order in range(1, max_considered_order+1):
    for candidate_list_of_voc in combinations(voc_list, current_order):
      intersection_size = voc_overlap(candidate_list_of_voc)
      union_estimate += relevant_alpha_vector[current_order-1] * intersection_size

  return union_estimate

def voc_inclusion_exclusion_naive_truncation(voc_list, max_order):
  assert max_order <= MAX_SUPPORTED_VOC_OVERLAP
  union_estimate = 0
  max_considered_order = min(max_order, len(voc_list))
  for current_order in range(1, max_considered_order+1):
    for candidate_list_of_voc in combinations(voc_list, current_order):
      intersection_size = voc_overlap(candidate_list_of_voc)
      union_estimate += (-1)**(current_order + 1) * intersection_size

  return union_estimate
```

B. Sequential Merge Method

The sequential merge method allows the system to calculate the de-duplicated reach across the publishers in sequence. Given N publishers' vector of counts: $c_1, c_2, \ldots, c_n$, where c_j is the vector of counts for the j-th publisher. The total counts of each respective vector of counts is given by: $m_1, m_2, \ldots, m_n$.

For each vector of counts from $j = 2$ to n, perform the following:

1. Estimate the overlap between c_1 and c_2:

$$\text{overlap} = \text{dotproduct}(c_1 - m_1/k, c_j - m_j/k)$$

2. Estimate the per-bucket overlap by allocating estimated total overlap to buckets:

$$\text{VoC(overlap)} = \text{overlap} * (c_1 + c_j) / (m_1 + m_j)$$

3. Construct the vector of counts of the union of c_1 and c_j:

$$\text{VoC(union)} = c_1 + c_j - \text{VoC(overlap)}$$

4. Set c_1 = VoC(union)

5. Increment j, repeat until j is greater than n.

This procedure can utilize an n pairwise de-duplication procedure. The formula for the VoC(overlap) can be changed as long as it captures the overlap between the two vectors of counts. There are several ways to determine the order of publishers for the sequential de-duplicaiton method. For example, the system can randomly pick the next publisher in the list, or pick the next publisher whose correlation with the currently selected publisher (e.g., c_j) is the smallest. This method can be generalized to perform the de-duplication in a hierarchical manor. For example, ((1, 2), (3, 4)), in which the system merges publisher 1 and publisher 2 into a single vector of counts, and merges publisher 3 and publisher 4 into a single vector of counts, and then de-duplicates both of those vectors.

C. Hierarchical De-deduplication

- *Notations:*
  - *m: number of publishers*
  - *n_i: marginal reach of publisher i*
  - *n_{ij}: intersection of 2 publishers i and j, which can be estimated by (the original) two-way VoC method*
  - *n_{123}, n_{1234}, , n1,2,,m: intersection of more than 2 publishers. Called the >2-way intersections.*
  - *k: number of buckets in VoC*
  - *v1v2v3: Dot-product of multiple vectors of the same length. For example:*

$$x \cdot y \cdot z = \sum_{i=1}^{n} x[i]\, y[i]\, z[i]$$

, where n denotes the vector length and x[i] denotes the ith element of x.

Using the above notation, the union of the multiset can be computed by the following formula:

$$\text{Union} \approx \sum_{i=1}^{m} n_i - \sum_{1 \le i < j \le m} n_{ij} + \sum_{1 \le i < j < k \le m} n_{ijk} - \cdots - (-1)^m n_{1,2,\cdots,m}$$

In the above formula, each two-way intersection n_{ij} can be estimated by the original two-way vector of counts method detailed herein. Intersections of more than two vector of counts can be computed by the following formulas.

$$n_{123} = (c_1 - n_1/k) \cdot (c_2 - n_2/k) \cdot (c_3 - n_3/k),$$

$$n_{1234} = (c_1 - n_1/k) \cdot (c_2 - n_2/k) \cdot (c_3 - n_3/k) \cdot (c_4 - n_4/k) - (n_{12}n_{34} + n_{13}n_{24} + n_{14}n_{23}) / k$$

$$n_{12345} = (c_1 - n_1/k) \cdot (c_2 - n_2/k) \cdot (c_3 - n_3/k) \cdot (c_4 - n_4/k) \cdot (c_1 - n_5/k)$$
$$- (n_{12}n_{345} + n_{13}n_{245} + n_{14}n_{235} + n_{15}n_{234} + n_{23}n_{145} + n_{24}n_{135} + n_{25}n_{134} + n_{34}n_{125} + n_{35}n_{124} + n_{45}n_{123}) / k$$

Generally, the formula to compute the intersection of *s* sets of user identifiers is given by the following formula.

$$n_{1,2,\cdots,s} = \text{Dot-product of } (c_1 - n_1/k),\ (c_2 - n_2/k),\ \cdots,\ (c_s - n_s/k)$$
- SUM(PROD(2-set partitions of all N pubs | each set in the partition includes ≥ 2 pubs)) / k
- SUM(PROD(3-set partitions of all N pubs | each set in the partition includes ≥ 2 pubs)) / k^2
- SUM(PROD(4-set partitions of all N pubs | each set in the partition includes ≥ 2 pubs)) / k^3
- ....

D. Direct Union Computation

The direct union computation formulas included below can directly estimate the union between multiple sets of user identifiers directly. The formulas below show the direct computation of the union, and utilize the same notation detailed above in section C.

$$\text{Union} \approx k + \sum_{i=1}^{m} n_i - (1 - d_1) \cdot (1 - d_2) \cdot \ \cdots \ \cdot (1 - d_m)$$

,

where d_i is the centered vector of counts of the i-th publisher (e.g., $d_i = c_i - n_i/k$).

E. Norms of the Total Sum Vector

This approach is a scalable and efficient method for computing the sum of intersections of each order of the hierarchical de-duplication described above in section C. The formulas in this section utilize the same notation described above in section C. The two-way formula included below.

$$\sum_{1 \leq i < j \leq m} n_{ij} = (|| \sum_{i=1}^{m} d_i \, ||_2^{\,2} \ - \sum_{i=1}^{m} n_i) \ / \ 2$$

,

where d_i is the centered vector of counts of the i-th publisher (e.g., $d_i = c_i - n_i/k$), and $||\ \ ||_2$ is the 2-norm of the vector. For example:

$$|| \, x \, ||_2^{\,2} = \sum_{i=1}^{n} (\, x[i] \,)^2$$

,

where $x[i]$ is the i-th element of vector x. The first formula above computes the sum of 2-way intersections, which can involve (m choose 2) = $O(m^2)$ terms, with linear complexity. If such formula can be extended to 3-way, then it can compute a sum involving $O(m^3)$ terms with linear complexity.

ABSTRACT

Systems and methods for determining the union of the set of user identifiers across multiple publishers are described. Each publisher computing device can use a list of hash functions to hash the respective set of de-duplicated user identifiers. Each publisher can assemble a vector of counts using the respective hashed set of user identifiers, where each coordinate in the vector of counts corresponds to a select of bit positions from the hashed set of user identifiers. Each publisher can add noise to each of the vector of counts to enhance the privacy of the system. Each publisher can transmit the respective vector of counts to a server to compute the union of the multiset without exposing any private or protected information about the user identifiers to any third-party. The server can compute the union of the sets described by the vectors of counts from each of the publishers using at least one of the methods described herein.