Problem statement and Value Proposition

For the past couple of years we have had many conversations about 1) how to run A/B tests and 2) how to deploy a feature to a small percentage of the user base.

Every time we need to segment users we end up having to write code on VCL and discuss whether we are segmenting per IP, per device or per “user”. When we want to launch a feature that might be disruptive, we normally follow the path of enabling it on a small wiki and extrapolating those results to a bigger wiki, which does not always work.

The goal of this document is to explore the design of a basic A/B testing framework such we do not end up having to modify our frontend caching code (varnish) every time we want to partially launch a specific feature. Our initial objective is to provide a simple framework to obtain data statistically valid in the absence of logged in users or sessions. Thus, in the initial design we are only considering readers using the Wikimedia sites anonymously. We do not consider editors.

Sample Use Case: Hovercards

We want to launch hovercards to 0.1% of the English Wikipedia user base and asses whether hovercards had an impact on our pageview metric. For the effect of this design document the code on hovercards is static. That is, we do not change the experiment while it is running.

Possible Implementations

Design Assumptions and Requirements
We are assuming a small number of experiments are run concurrently (about 50) whether they overlap or not is not necessary from a statistical standpoint although it might be desirable from an experiment management standpoint to better be able to reason about experiments.

We also assume that the only dimension on which we are assigning experiments are “users” (or rather “devices”), not articles. In A/B testing speak this means that our unit of diversion is the user/device. This means that our tests can be run over the user base (deploy feature X to 1% of devices) but not to articles (deploy feature X to 1% of articles that have more than 1000 words).

Double Bucket

Instead of using a cookie with a unique token, we use a cookie that can take on N different values where N << # of users.

Users will be “approximately” randomly assigned to an experiment.

Plain English

1. User Request Comes in to varnish.
2. We assign a long-term binning bucket [0,1Million] via a cookie[1] with a long expiry to every request. Hence, every bucket comprises 0.001% of user base.
3. We randomly assign a subset of buckets to each experiment from the set of buckets not in use by any other experiment. A test running on 0.1% of users (0.05% control, 0.05% test) will need 100 free buckets.

Statistical Properties

By letting N be fairly large, randomly assigning users to buckets and then randomly assigning buckets to experiments, we can approximate random assignment of users to experiments.
The more segments/buckets there are the better the approximation to random assignment of users. Over time, the group of users in one bucket will be systematically different from the group of users in another bucket as a result of being in different experimental conditions in the past. We want to make sure that for a new experiment, the populations of users in each experimental condition are roughly the same. This is traditionally done by randomly reassigning individual users to each new experiment, which we cannot do, due to the lack of a unique token. By randomly assigning a large number of buckets (say 1000) to each experimental condition we

[1] Global cookies are not possible on our system but we could have cookies that span large projects like *.wikipedia. The domain in which we most commonly run A/B testing is *.wikipedia.

can approximate full random assignment of users. We are assigning 1000 buckets per experiment over a ‘population’ of 1 million buckets.

This means that N should be no smaller than:

1000 * (1/min(fraction of users in experiment))

A more serious downside of the Double Bucket design is that we cannot compute user level metrics and properly compute confidence intervals or evaluate statistical hypothesis tests (for more details, see here).

Pseudo-Code

Binning and A/B testing info get published to x-analytics together with pageview data, below is an example of how would this code look in varnish.

Example:
Feature X is to be tested in 0.1% of user base, needs random 1000 buckets, half of which are control. User buckets are “draw” at random from buckets that are not already in use.

```
experiments => { # 1,000,000 total bins to use: 0-999,999
    'FeatureX/A' => [1,17678, 56,90,....], # array with random 50 elements that does
not include already used buckets
    'FeatureX/B => [5, 999, 70980, 8905...], # array with random 50 elements 0.5% of
all devices
}

receive_frontend_request {
  // Static code:

  unsetHeader("X-Weblab");

  if (hasCookie("SampleBucket")) {
    // we'll also validate that the value looks sane, and
    // treat it as non-existent if it's not valid
    Bucket = getCookie("SampleBucket")
  }
  else {
    Bucket = rand(0,1000000);
    setResponseCookie("SampleBucket", Bucket);
  }

  // templated out automatically from 'experiments' data above:
  if (Bucket in FeatureXA_Set) {
    setRequestHeader("X-Weblab:", "FeatureX/A");
  } else if (Bucket in FeatureXB_Set ) {
    setRequestHeader("X-Weblab:", "FeatureX/B");
  }
}
```

Privacy Properties

A bucketing solution is only more privacy preserving than a token, if the buckets are large enough to contain a large number of users. If we assume our user base is about 1000 million (1 billion of distinct devices)[2] and we have about 1 million buckets every bucket has 1000 users. The idea is that such a set would have a k-anonymity of 1000 (1000 is just an example, number could be higher, depends on the granularity we want in our experiments). The issue we have is that this level of anonymity is only real in *large enough wikis.* For small wikis, say, tagalog wikipedia, it is likely that a user in bucket number, for example, “23” is the *only* user for that wiki on that bucket, thus, de facto turning the bucket (regardless of whether there are 999 more users on it) on a unique token for that wiki.

This proposal could only be used in wikis whose reader numbers are large enough to guarantee an acceptable level of anonymity. For more details on why unique tokens are not acceptable see the many discussions and concerns on our community about privacy. The gist of it is that unique tokens allow to follow the browser session of a user and to our community that is unacceptable privacy invasion.

It is also worth mentioning that we could use the double bucketing binning to implement a “progressive roll out of features” that would work for all clients, regardless of javascript support and could be used to roll out features on first pageload.

Other Shortcomings

Issues with expiration of cookies and Nondeterminism of assignments

The SampleBucket cookie is a long expired cookie. We want to avoid the issue with users coming in and out of bins at the 30 day mark. Now, cookies, in the internet do not last forever. So results from this system are subjected to be more imprecise than binning that might come from using user’s info (like an e-mail) to create a hash that is later assigned to an experiment. Our assignment from device to experiment is not deterministic. An assignment that relies on user information however will be.

Unique Token in Local Storage + Restricted Logging

We set a persistent unique token in each client’s local storage. Each experiment provides a unique name and clients are assigned to different treatment conditions by hashing the token along with the experiment name into different buckets. Note that this imply that readers might be in two experiments at any one time.

[2] We have estimated our Monthly Active Unique Devices to be around 1 billion (deduplicated across projects) but this is a rough estimate we have not published externally. We expect to have data in this regard by July 2017.

This is similar to how we are running experiments right now with one fundamental difference. **The schema doesn't include any information other than the one it pertains to the experiment There is not even a capsule that wraps the event data with additional information such as user-agent.**

Example:

ExperimentSchema:
Token: (string) the unique token assigned to the client
Condition: (enum) the experimental condition the client is in
Event: (enum) the type of event the client experienced or triggered
Result: (Float) a number associated with the event client experienced or triggered

To evaluate a fundraising experiment, for example, we could log both "impression" and "donation" *events*. For donation events, we can log the amount donated in the *result* field.

Statistical Properties

This proposal has all the right properties from a statistical perspective.

1. The sample population is restricted to clients with local storage
2. Clients are randomly assigned to treatment conditions at the start of every experiment
3. Clients remain in their treatment group for the entire duration of the experiment (unless they reset local storage)
4. We can use metrics over client level aggregates to ensure that we don't violate independence assumptions of statistical tests used for interpreting experimental results
5. Experiment can be run concurrently (with some small caveats)
6. We can run experiments that require measuring small effects or rare events.

Privacy Properties And Privacy Issues

1. Since the token is in local storage and not a cookie, we can control what kind of information is sent along with the cookie. So for example, we avoid sending a unique token along with every webrequest. It is worth mentioning that localStorage, just like cookies is restricted to cross domain restrictions.
2. The **ExperimentSchema** ensures that the token is not *logged* with any PII data such as IP, UA, requested article, etc. It also ensures that the token is not *sent* with any PII data (except IP and UA, which is unavoidable). We had prior discussions on why we do not log tokens with PII data [6]. A notable exclusion regarding tokens are mobile apps, as appinstallIDs are sent with every request if user has opted in.
3. Note that as described the system would also work for apps, the appInstallID is already playing the part of the token. Now, in the case of apps ids are sent with every request

thus using the same appInstallID for experiments reduces the privacy of users even further

Other Shortcomings

Javascript enabled only
This technique is eventlogging based which implies javascript enabled clients on the web.

Appendix:

Proper Randomization when unit of diversion is a cookie.

In order to understand why proper randomization of events is a concern there are two definitions we need: "unit of analysis" and "unit of diversion". "Unit of analysis" is the level at which conclusions are being pitched and "unit of diversion" is the level at which we collect the data. An easier way to remember these concepts is that the unit of diversion is the denominator of your metric.

This is easier explained with an example. Let's say we are measuring where feature X increases/decreases clickthrough rate. If we measure # of clicks per pageview our unit of diversion and analysis is a "pageview". We are sending pageviews to treatment A or B and measuring clicks on each treatment. We assume every event we are measuring is independent for every other event and in that sense every event is a "random draw".

Now, if we measure number of clicks using user ("device" or "cookie", really) as our unit of diversion (i.e. we are collecting clicks tagged with a cookie that identifies whether the user is on treatment A or B) our unit of diversion is a cookie.

In the first case we can assume that events for which we are collecting clicks are independent, our collection is event-based. In the second case events for which we collect data are not truly random as events "can" be grouped by device (as a proxy for a user) and thus they might not be truly independent from each other.

The cookie based collection has statistical value if we assume the variability of our metric will be higher and also that it might need to be calculated empirically. That is, we cannot make any assumptions as to the shape of the distribution of our data.

To understand the level of variability of the metric we choose to use, for example, clickthrough it will be useful to run an A/A test before we run an A/B test to estimate thresholds of variability.

For more details see: [3] 5.2 section, Experimental design.

Combined Approach

Progressive Rollout

It is worth mentioning that we can use some of the ideas discussed in AB testing to implement progressive rollout of features. If we want to hit 1% of our user base with an initial feature launch we will need to distribute 100 buckets server side. Small enough wikis can be assigned a bucket number and in large enough wikis we can do random assignation of buckets per user.

This would not collide with the cookie/local storage AB testing scheme but will be an additional cookie send on each page request.

References and background material

[1] Designing and Deploying Online Field Experiments:
http://hci.stanford.edu/publications/2014/planout/planout-www2014.pdf
[2] PlanOut: https://facebook.github.io/planout/
[3] Overlapping Experiment Infrastructure: More, Better, Faster Experimentation:
http://static.googleusercontent.com/media/research.google.com/en/us/pubs/archive/36500.pdf
[4] Seven Pitfalls to Avoid when Running Controlled Experiments on the Web:
http://ai.stanford.edu/~ronnyk/2009-ExPpitfalls.pdf