Trustworthy Experimentation Under Telemetry Loss

Jayant Gupchup
Microsoft
jayagup@microsoft.com

Yasaman Hosseinkashi
Microsoft
yahossei@microsoft.com

Pavel Dmitriev
Outreach.io
pavel.dmitriev@outreach.io

Daniel Schneider
Microsoft
danielsc@microsoft.com

Ross Cutler
Microsoft
rcutler@microsoft.com

Andrei Jefremov*
Unaffiliated
andrei.d.jefremov@gmail.com

Martin Ellis
Microsoft
maellis@microsoft.com

ABSTRACT

Failure to accurately measure the outcomes of an experiment can lead to bias and incorrect conclusions. Online controlled experiments (aka AB tests) are increasingly being used to make decisions to improve websites as well as mobile and desktop applications. We argue that loss of telemetry data (during upload or post-processing) can skew the results of experiments, leading to loss of statistical power and inaccurate or erroneous conclusions. By systematically investigating the causes of telemetry loss, we argue that it is not practical to entirely eliminate it. Consequently, experimentation systems need to be robust to its effects. Furthermore, we note that it is nontrivial to measure the absolute level of telemetry loss in an experimentation system. In this paper, we take a top-down approach towards solving this problem. We motivate the impact of loss qualitatively using experiments in real applications deployed at scale, and formalize the problem by presenting a theoretical breakdown of the bias introduced by loss. Based on this foundation, we present a general framework for quantitatively evaluating the impact of telemetry loss, and present two solutions to measure the absolute levels of loss. This framework is used by well-known applications at Microsoft, with millions of users and billions of sessions. These general principles can be adopted by any application to improve the overall trustworthiness of experimentation and data-driven decision making.

KEYWORDS

Online controlled experiments, AB testing, client experimentation, telemetry loss, data loss, experimentation trustworthiness

ACM Reference Format:
Jayant Gupchup, Yasaman Hosseinkashi, Pavel Dmitriev, Daniel Schneider, Ross Cutler, Andrei Jefremov, and Martin Ellis. 2018. Trustworthy Experimentation Under Telemetry Loss. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3269206.3271747

1 INTRODUCTION

AB testing has helped organizations evaluate new ideas, tune parameters, catch critical bugs, predict infrastructure needs, measure customer value, and help with team planning [6, 14, 22]. In the simplest controlled experiment, users are randomly assigned to one of two variants: control (A) or treatment (B). Typically, the control is the existing system, and the treatment is the existing system with a new feature *X*. If the experiment was designed and executed correctly, the only difference between the two variants is *X*, establishing a causal relationship between the change made to the product and changes in user experience. The ability to derive this causal relationship is a key reason for widespread use of controlled experiments. For example, online experimentation helped Bing identify dozens of revenue-related changes to make each month, collectively increasing revenue per search by 10-25% each year [13].

Although online experimentation is well known in web services, it is also used in mobile and desktop applications [4, 23]. Experimentation is especially useful for mobile apps, since they are used by a diverse set of devices and in a wide variety of network conditions. Such heterogeneity cannot be fully reproduced in a lab, limiting the extent to which applications can be tested internally. We refer to such experimentation scenarios as *client experiments*, to distinguish them from experiments that purely impact server-side behavior.

One important aspect of client experiments is that the telemetry data collection is done over the Internet, with clients uploading experiment data to cloud services. This introduces several possible sources of failure, with telemetry data potentially being delayed, lost, or collected at different rates among variants (e.g., due to bandwidth limitations or software bugs). Moreover, since incoming telemetry data are typically processed in complex data pipelines, there are further opportunities for bugs to be introduced, resulting in missing experiment data; note that all online experiments are susceptible to this risk, not just client experiments. The key observation is that when telemetry loss is not uniform at random, experiments can be exposed to major population bias, increasing the risk of incorrect conclusions.

[*] The work was conducted while Pavel Dmitriev and Andrei Jefremov were at Microsoft.

CIKM '18, October 22–26, 2018, Torino, Italy
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3271747

To help illustrate the direct impact of telemetry loss on experimentation, we will use an example from one of our recent experiments. This experiment (which we will refer to as the *ui-change* experiment) involved evaluating the impact of a change to the user interface used to survey user satisfaction in Skype, one of our communication products. After collecting data from the experiment, we found a surprisingly large improvement of 8% in user ratings. However, further investigation revealed a software bug, where clients in the treatment group failed to submit poor ratings, resulting in a 13% difference in telemetry loss compared to the control group. After fixing the issue and repeating the experiment, we found no statistical difference between the interfaces.

In the previous example, telemetry loss was fairly easy to detect; however, there are also more subtle cases, where the impact may not be so obvious. For example, in cases where the treatment has different effects across the overall population, this can combine with non-uniform telemetry loss across the population to obscure the results of the experiment. Specifically, telemetry loss can result in under-representation of substantial segments of the population where the treatment has effects, leading an experimenter to incorrectly conclude that there is no overall difference. An analogy to this situation can be found in political polling, where polling firms might under-sample certain segments of the voting population; by surveying a non-representative subset of the electorate, they miss important shifts in voting intention and therefore make incorrect predictions about election results [1, 10].

Our experiences clearly demonstrate the importance of reducing telemetry loss; however, eliminating this loss completely (although desirable) is not a realistic goal for large-scale client applications. This demands a robust methodology to ensure the trustworthiness of experimentation and data-driven decisions made in the presence of such losses. In this paper, we describe the methodology and tools developed at Microsoft to achieve this goal. Considering that the impact of such loss is amplified in client experiments, we use specific in-depth examples from two widely used applications; *Skype*, a popular communication application, and *OneNote*, a widely used application for free-form note taking and collaboration. Note, however, that the proposed solutions are generic and can be applied to any online experimentation system (client- or server-based). To the best of our knowledge, our work is the first to address telemetry loss in online experimentation. The main contributions of this paper are as follows:

- A taxonomy of telemetry loss scenarios and best practices to minimize the loss.
- Examples of real experiments conducted at our organization illustrating the biases introduced by data loss.
- A theoretical breakdown of biases caused by data loss and their properties in practice.
- A methodology to simulate experiment results under no loss.
- A framework to estimate how much data loss can be tolerated by the experimentation system for trustworthy operation.
- Two methods for measuring data loss and clear guidance on their application.

The rest of the paper is organized as follows: Section 2 provides detailed background and related work. Section 3 outlines a systematic framework for characterizing loss and estimating the level that can be tolerated by an experimentation system. Section 4 describes and evaluates two novel approaches for measuring telemetry loss, and recommends ways to reduce it. Section 5 shares some practical lessons for the benefit of the experimentation community, and Section 6 summarizes the paper and discusses future work.

2 BACKGROUND

2.1 Related Work

Online controlled experimentation is an active research area, focused on topics such as running controlled experiments in practical settings [5, 11, 12], and new statistical methods to improve metric sensitivity [3]. For an in-depth introduction to online controlled experiments, see [4, 19]. These studies provide context for our work, sharing algorithms, methods, and lessons learned from running real experiments in practice. Specifically, telemetry loss was listed as a common reason for misinterpreting the results of experiments [5].

A common technique to measure packet loss in the Internet is through the use of *sequence numbers* (e.g., in the RTP protocol [18]). The sequence number is incremented for each data packet sent, allowing the receiver to detect packets that do not arrive. We apply a similar approach (referred to as the sequence method) to measure data loss, which we describe in Section 4.2. The challenges in mirroring data between unreliable endpoints have been studied in detail by the database community [7, 21]. While telemetry gathered from our apps do not require transaction guarantees, mobile environments have constraints in terms of network reliability and local storage. Moreover, most analysis systems do not tolerate delays of more than a day. Nevertheless, client apps can borrow ideas such as lazy replication to improve telemetry reliability, especially if the experimentation and analysis system can tolerate longer delays.

Although telemetry data loss is recognized as an important problem, there is little prior research discussing data loss in the context of online experiments. A similar problem of missing treatment values in behavioral randomized experiments is discussed in [16]. This method assumes no data loss for outcome variables but possible loss of received treatment indicators. Our study focuses on the situation where outcome variables and possibly covariates are lost but the treatment assignment is known. A methodology to identify treatment effects under data loss for categorical outcome metrics in the context of voting experiments is described in [8]. This study is relevant to our work, and captures one of the biases characterized in this paper; however, we formalize the different types of biases induced by data loss, and provide a general framework to establish a tolerance threshold for AB test systems operating at scale.

2.2 Overview of client telemetry loss

The processing of client telemetry data collected from apps is a complex and tedious process. In order to highlight these complexities, we first present a high-level overview of a typical telemetry flow. Figure 1 shows the interaction between the client and server components and the telemetry flow within an application. We refer to each client-server interaction as a session. While we use *Skype* to highlight the details of the complexities, many of the issues are general and apply to *OneNote* and other client applications.

In *Skype*, telemetry data (referred to as events) for a call (or session) are collected at each participating client and at the server.

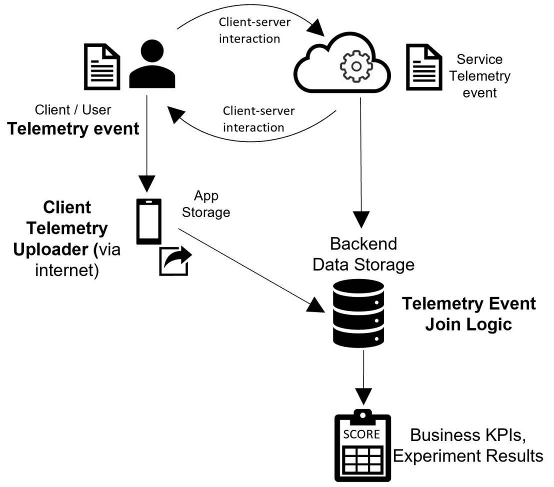

Client-server interaction
Service Telemetry event
Client / User
Telemetry event
Client-server interaction
Client Telemetry Uploader (via internet)
App Storage
Backend Data Storage
Telemetry Event Join Logic
SCORE
Business KPIs, Experiment Results

Figure 1: Telemetry flow for the client and server components in a typical application.

These events are sent to the backend store using an uploader application. The server and backend storage are typically co-located in the cloud so the network connection is reliable and buffering capacity is not a constraint. The client events are cached locally and transmitted to the backend store opportunistically by the uploader. Once events arrive at the backend, they are combined to form a call record (or session record) table, which is used for tracking business metrics and experimentation analysis. Next, we will look at each component and the sources of telemetry loss in more detail.

Telemetry Events: In an application, events are reported by a number of internal components interacting with one another. For instance, in *Skype*, the call signalling component and service telemetry provide information about call establishment, the audio/video (AV) components provide telemetry about AV quality, and finally, the UI layer reports the overall quality of experience as reported by the user. This user rating event (previously discussed in reference to the *ui-change* experiment) is not collected for every session, but is available only for a subset of randomly selected calls, when users are asked to rate their experience. In contrast, call setup telemetry (henceforth, *CST*) is collected for every call attempt. The *CST* event is used to compute key metrics such as call establishment rate and call duration. It is worth mentioning that every event also carries experiment configuration information (i.e., treatment assignment identification). This information is used to generate *experiment scorecards*, lists of metric comparisons to help track the impact of treatments on user experience, as well as metrics for experiment and data health. Typically, each component generates one event per session, which are cached locally and uploaded to the backend using an uploader application. Henceforth, the details within an event will be referred to as “measures”.

Storage and Event Uploader: Challenges faced by the telemetry uploader application, which can result in data loss, include:

- Bandwidth heterogeneity: Clients make calls (or initiate sessions) in a range of network conditions, including metered 2G networks. These bandwidth constraints result in poor event transmission reliability.
- Sharing bandwidth with the service: Sending telemetry during a session can impact the session experience, so the telemetry system needs to be service-aware, and back off to prioritize smooth functioning of the app.
- App termination: Once an app has been moved to the background, it can be abruptly terminated by the operating system or the user. This prevents some state from being persisted to disk.
- Limited storage budget: Typically, the local cache is implemented as a buffer with a limited storage budget (e.g., on low-end mobile devices). If the telemetry queue exceeds the allocated storage, events are dropped.
- Event prioritization: Events are prioritized to transmit the most critical information first. Business KPIs (e.g., user ratings) are sent first, which may cause starvation of lower priority events (e.g., technical metrics). Typically, only the highest priority events are sent reliably (with re-transmission enabled) while the lower ones are not. It is worth noting that many component-level experiments rely on metrics carried by lower priority events.

As shown in Figure 1, events are collected from both client and server, which upload events independently. This can result in telemetry events from the same session arriving at different times, and suffering different rates of data loss.

Event Joins: Once events are uploaded to the backend system, they are joined together to form the call record (or session record) that powers experimentation analysis. Typically, the join key is $\{session\_id, endpoint\_id\}$. Since the events are uploaded separately, we may receive some events, while others may fail or suffer delays. If a component fails to record *endpoint_id* or *session_id* correctly due to software bugs, those events will be dropped downstream due to the join failure. In the *Skype* scenario, each call has a unique ID and the join key is $\{call\_id, endpoint\_id\}$

2.3 Biases caused by telemetry loss

In this section, we describe real experiments in *Skype* and *OneNote*, illustrating different types of bias caused by telemetry loss. These experiments were run at scale having more than a million observations in each variant.

***OneNote* First-Run Experience - Differing loss rates within segment:** This experiment tested the number of swipe screens after the installation of the app (referred to as the “first-run experience”). The control experience showed the user three swipe screens, while the treatment only showed one. The hypothesis was that three swipe screens was too many for some users, and that reducing it to one would yield higher engagement with the app. The AB scorecard for the first-run population showed a statistically significant difference in the number of users in treatment and control, a phenomenon known as sample ratio mismatch (SRM) [5]. The SRM was found to be around 4%, with more users in the treatment compared to the control. Furthermore, within the population of users running the app for the first time, the scorecard suggested that the users in the treatment group had lower engagement. The overall scorecard computed on the entire population, on the other hand, did not show an SRM, showing instead that users in the treatment group had higher engagement with the app. The SRM in the first-run scorecard was caused due to a bug in the control group, resulting in a failure to submit telemetry in cases where users abandoned the first-run screen. In this case, telemetry loss led to a biased estimate of the engagement metric in the scorecard. After the bug was detected and fixed, the experiment was repeated, showing no improvement in either the overall population or the first-run population.

Skype **Video Bandwidth - Differing loss rates among variants:** This experiment aimed to improve video quality in low bandwidth conditions. It had four treatments, each testing a different setting. We found that treatment groups with a lower bandwidth threshold were exposed to higher data loss rates due to more challenging conditions. The data loss rate in each setting was found to be statistically different; the difference in data loss between the variants with the highest and lowest loss rates was 0.7%. In this experiment, the data loss of the event carrying the outcome metric was directly correlated with the treatment itself, (i.e., the treatment has an impact on the data loss rate). The unbalanced loss rates between treatment and control variants resulted in a biased comparison, ultimately invalidating the results of the experiment.

Skype **Video Decoder - High absolute loss rates across variants:** This experiment focused on improving video quality for mobile clients. Initial analysis did not show any significant difference between control and treatment, and there were no apparent problems with data quality. However, further investigation showed that the video telemetry had a loss rate over 10%, since the video telemetry event had a lower priority (as discussed in Section 2.2), and suffered the highest loss rates compared to other client events. Since none of the outcome metrics had changed between treatment and control, it was unclear whether "no change" was due to telemetry loss or truly due to the treatment having no impact. From this experiment, we learned that, even when the loss rate between control and treatment is balanced, a very high overall loss rate can lead to massive bias in outcome metrics, masking the true impact of the treatment on user experience. In such cases, the scorecards show an incomplete picture about the effect of the treatment, and can lead to inaccurate conclusions. This topic will be discussed in more detail in Section 3.

Skype **Headset Impact - Unable to construct segments:** In this experiment, we were interested in improving the audio experience of users using headsets. The outcome metric was the average user rating, which comes from a reliable event with low loss rate. However, for each variant, we also needed to identify whether a headset was used during the call, which relies on device usage information reported from a low priority event with high loss rate. As a consequence, analysis of this experiment was biased due to the loss of the information needed to identify the sub-populations of interest.

The examples and scenarios presented above are representative of client experiments. As a result, it is critical to consider telemetry loss of events at the design phase of the experiment. Our approach was to look at this problem in a top-down fashion. We first begin by evaluating the impact of loss on experimentation to understand how much loss can be tolerated. Then, we address the challenge of measuring the absolute level of loss for each event to assess how far we are from our desired loss target. As mentioned before, our solutions are general and apply to any experimentation system; we use the context of *Skype* to convey these ideas more concretely since this app is used in challenging environments.

3 EXPERIMENTATION UNDER DATA LOSS

The presence of telemetry loss in experimentation data raises some natural questions. In particular,

(1) How different would the current experiment results be if there was no telemetry loss?
(2) How much telemetry loss can our experimentation system tolerate, while still providing trustworthy results?

Addressing the first question is crucial to ensure the correctness of decisions made by experiments impacted by telemetry loss. By answering the second question, we can determine a practical goal for improving telemetry loss. To quantify the impact of telemetry loss, we develop a model to formulate the biases caused by loss on AB test results and explore its practical implications. Then, we present a practical algorithm to simulate the results of an experiment under no telemetry loss. Finally, present a general algorithm to estimate a threshold for telemetry loss that can be tolerated in a trustworthy experimentation system. This algorithm can be adapted and tuned for different experimentation platforms with variable sensitivity to telemetry loss.

3.1 Statistical Model for Data Loss Impact

The change in user experience caused by the treatment is commonly referred to as "treatment effect" in AB testing. Without loss of generality, we describe the mathematical models in this section for a simple AB test with two variants: control and treatment. Following the linear model framework for randomized experiment [17], if no blocking variable is present, then treatment effect (β_T) is modeled as the difference in expected outcome variable Y conditioned on treatment T (an indicator variable representing treatment):

$$\begin{aligned} Y &= \beta_0 + \beta_T T + \varepsilon && (1) \\ \beta_T &= E(Y|T=1) - E(Y|T=0) && (2) \end{aligned}$$

assuming ε has mean 0 and finite variance σ^2.

In this framework, $\beta_0 = E(Y|T=0)$ represents the baseline or current average of the outcome metric while β_T measures its expected change if treatment is applied to all population. Under fairly general assumptions, the difference between the averages of Y in treatment and control,

$$\Delta = \bar{Y}_1 - \bar{Y}_0 \tag{3}$$

is an unbiased estimator of β_T and its normalized value $\frac{\Delta}{se(\Delta)}$ offers asymptotic unbiased significance tests. This is widely used in online and client experimentation as a reliable decision making mechanism. However, this solution is no longer adequate in the presence of telemetry loss, where model (1) needs to be extended by a new term to capture the loss. Let $L \sim Bin(p_{T,X})$ be 1 whenever outcome variable Y is lost and 0 otherwise. Then model (1) under data loss extends to:

$$Y = \beta_0 + \beta_T T + \beta_L L + \beta_{int} T \times L + \varepsilon \tag{4}$$

According to (4), $E(Y|_{T=1}) = \beta_0 + \beta_T + \beta_L E(L|_{T=1}) + \beta_{int} E(L|_{T=1})$ and $E(Y|_{T=0}) = \beta_0 + \beta_L E(L|_{T=0}))$. Therefore,

$$E(\Delta) = \beta_T + bias(\Delta); where$$

$$bias(\Delta) = \beta_L \times [E(L|_{T=1}) - E(L|_{T=0})] + \beta_{int} E(L|_{T=1}) \tag{5}$$

The two additive terms in (5) are the two types of biases with very different characteristics in practice: *correlation bias* and *interaction bias*.

Correlation Bias (corr-bias): $\beta_L \times (E(L|T = 1) - E(L|T = 0))$, measures the correlation between data loss and treatment. This is non-zero when the treatment changes the behavior of the app in sending telemetry, or causes an indirect change in the loss distribution. For example, the *Video Bandwidth* experiment described in Section 2.3 exhibits corr-bias.

Interaction Bias (int-bias): $\beta_{int} E(L|T = 1)$, measures the interaction between treatment effect and data loss. This occurs when the treatment effect is expected to be different under $L = 1$ and $L = 0$. For example, the *Video Decoder* experiment described in Section 2.3 exhibits int-bias.

Under the missing at random (MAR) assumption, L is independent of T and Y; i.e., $E(L|_{T=1}) = E(L|_{T=0})$ and $\beta_{int} = 0$. Hence, both bias terms would be zero and observed delta in Formula (3) still provides an unbiased estimator of true impact. The only cost of telemetry loss in this case is reduced sample power. In practice, however, we observed that telemetry loss, even when occurring at the same rate for treatment and control group, is not independent of the outcome variable or treatment itself.

In the *Video Bandwidth* experiment, more challenging bandwidth conditions led to a higher loss rate. This resulted in a positive correlation between data loss and treatment, for video metrics. Since the data loss is not at random and is more concentrated on poor user experiences (i.e., low bandwidth is closely related to poor user experience [9]), there was no way to separate the impact of treatment from the inherent differences in samples. Therefore, the video metrics reported in our scorecards were inconclusive.

Correlation bias manifests itself as statistically different loss rates between treatment and control samples, so it is fairly easy to detect. However, there is no easy way to recover from it, since changes in the outcome metrics could be attributed to either the treatment change or the telemetry loss. If the correlation bias is a result of a bug, such as in *ui-change* example, the solution is to fix the bug and run the experiment again. However, if the correlation bias is because of the nature of treatment, such as in *Video Bandwidth* experiment, the only option is to analyze the experiment using other metrics that are not impacted.

In contrast, interaction bias is difficult to detect and correct for. This hidden bias can easily mislead experimenters who observe the same rate of telemetry loss on both treatment and control side and conclude there is no obstacle to analyzing the experiment data.

3.2 Challenges in Correcting Data Loss Bias

Covariate post-stratification is a common statistical technique to adjust estimates of an outcome variable by re-weighting [15]. This method is especially beneficial in handling data loss if there is a random variable X that is strongly correlated with loss, and the true distribution of X is observed. This adjustment can be achieved by dividing the population into strata with known weights, imputting the lost values using the correlated feature, and finally re-computing the outcome metric using the strata weights.

In practice, applying this method is problematic and finding a reasonable covariate is non-trivial. In *Skype*, when data loss occurs due to extreme network conditions, obtaining accurate network estimates is challenging, and therefore correlated covariates are not easily available. Moreover, data loss is a nonlinear function of multiple variables, as discussed in Section 2.2. Another limitation of this method is faced when one or more strata are completely lost, such as application crashes which lead to 100% data loss. In such cases, knowledge of loss rates does not help to correct for the bias. In the absence of suitable covariates, we have developed solutions to estimate boundaries of data loss bias instead of correcting for it.

Algorithm 1 Simulate Overall Treatment Effect

```
Input: $[\bar{y}''_{ctrl}, s''_{ctrl}, \beta_{int}], [\bar{y}'_{ctrl}, s'_{ctrl}, s'_{trt}, \bar{y}'_{trt}], [l_{ctrl}, l_{trt}]$
Output: overall treatment effect under no loss: $\frac{\Delta}{se(\Delta)}$
1: Estimate sample mean for lost data points according to (9)
2: Estimate overall delta:
       $\Delta = (1 - l_{ctrl}, 1 - l_{trt}) \times \begin{pmatrix} \bar{y}'_{ctrl} \\ \bar{y}'_{trt} \end{pmatrix} + (l_A, l_B) \times \begin{pmatrix} \bar{y}''_{ctrl} \\ \bar{y}''_{trt} \end{pmatrix}$
3: Calculate overall variances $s^2_{trt}$ and $s^2_{ctrl}$ according to (8)
4: Calculate overall treatment effect $\frac{\Delta}{se(\Delta)}$ where $se(\Delta)$ represents
   the standard deviation of delta
```

3.3 Estimating Boundaries of Data Loss Bias

In this section, we address how the results of experiments would be different if telemetry was not lost. Simulation is a powerful tool for this, since it does not require external data or covariates correlated with loss L. Beyond observed summary statistics, we need only the measured loss rates for each sample and a range of *scenarios*.

A *scenario* characterizes an intuition about the lost data points, such as "lost events are correlated with poor user experience, but the experience is improved by the treatment" or "lost events are correlated with poor user experience, and the treatment degrades the experience even further". The idea is to impute the lost data points under certain scenarios and then reconstruct the overall treatment effect. This can be done by decomposing the mean and standard deviation of a complete sample $(\bar{y}, s)$ into observed and unobserved pieces. We use $(\bar{y}', s')$ and $(\bar{y}'', s'')$ to refer to the summary statistics of observed and lost parts, respectively. Here is the breakdown for overall sample mean $\bar{y}$ and variance s^2:

$$\begin{aligned}
\bar{y} &= (1 - l)\bar{y}' + l\bar{y}'', \mathit{and} && (6)\\
s^2 &= \frac{1}{n}\sum_i (y_i - \bar{y})^2 \\
&= \frac{1}{n}[\sum_{i \in obs} (y_i - \bar{y})^2 + \sum_{i \in lost} (y_i - \bar{y})^2] \\
&= \frac{1}{n}[\sum_{i \in obs} (y_i - \bar{y} \pm \bar{y}')^2 + \sum_{i \in lost} (y_i - \bar{y} \pm \bar{y}'')^2] \\
&= (1 - l)s'^2 + ls''^2 + (1 - l)(\bar{y}' - \bar{y})^2 + l(\bar{y}'' - \bar{y})^2 && (7)
\end{aligned}$$

where $l = \frac{n'}{n}$ is the sample loss rate. By replacing $\bar{y}$ according to (6), equation (7) simplifies to:

$$s^2 = (1 - l)s'^2 + ls''^2 + l(1 - l)(\bar{y}' - \bar{y}'')^2 \tag{8}$$

This breakdown holds for both treatment and control samples. We will refer to them by *trt* and *ctrl* suffixes.

Each scenario must specify three unknown parameters: $\bar{y}''_{ctrl}$, s''_{ctrl} and β_{int}. The first two parameters are our hypothesis about the status of lost data points, regardless of experimentation. For all simulations, we estimated these summary statistics from the bottom 10th percentile of quality metric distributions. This choice was motivated by our observation that most data loss is associated with poor experiences. β_{int} represents our assumption about the impact of the treatment on lost data. This is established on a per-case basis for each experiment, using domain knowledge obtained from lab or offline results. Large absolute values for β_{int} are indicative of experiments with high sensitivity to data loss. Setting β_{int} to zero implies there is no difference in treatment effect on unobserved and observed samples, hence there is zero sensitivity to data loss.

Assuming that treatment may only change the metric baseline and not its variance, i.e. $s''_{ctrl} = s''_{trt}$ the three input parameters specified by a scenario are sufficient to reconstruct the overall treatment effect. Using model (4), $\bar{y}''$ can be estimated by:

$$\bar{y}''_{trt} = \bar{y}''_{ctrl} + \delta' + \beta_{int} \tag{9}$$

under scenario $[\bar{y}''_{ctrl}, s''_{ctrl}, \beta_{int}]$ where $\delta' = \bar{y}'_{trt} - \bar{y}'_{ctrl}$ is the observed delta. The simulation process is a simple application of (6), (8) and (9), as described in Algorithm 1.

If needed, the equal variance assumption can be simply relaxed by adding s''_{trt} an extra input parameter to Algorithm 1 and applying it when calculating $se(\Delta)$. This parameter can be initiated using historical values from previous experiments. We continue with equal variance assumption since it is verified for our application.

The result of applying Algorithm 1 is a new scorecard with simulated values under no loss. If there is a large difference between the simulated and observed scorecards, this implies that the data collected from the experiment are inconclusive. On the other hand, the observed scorecard is deemed to be trustworthy enough if it is not practically different from the simulated scorecards.

Table 1 shows an example of the output of Algorithm 1, for the *Video Decoder* experiment under two different scenarios, along with observed statistics for a few metrics. The experiment data used for this simulation include approximately 30 million observations for each variant. As discussed in Section 2.3, the video telemetry in this experiment had over 10% loss. According to the observed scorecards, none of the metrics showed significant change. The video-related metrics (shown in *italic*) are calculated using high loss rate events, while the others are based on low loss rate events. As expected, the latter set of metrics are the same in observed and simulated scorecards. However, the video related metrics show wildly different results. Under the worst-case scenario ($\beta_{int} = 5\%$), overall VideoFreezeDuration shows a 2.43% increase, while under the best-case scenario ($\beta_{int} = -5\%$) it shows a 2.47% decrease. This indicates a large confidence interval for true effect, that spans over a diverse set of conclusions from "treatment improves user experience" to "treatment degrades user experience". In such cases, we flag the experiment as inconclusive.

	Relative Delta (P.Value)		
Metric	**Observed**	**Best-case**	**Worst-case**
DurationMean	-0.02% (0.76)	-0.02% (0.76)	-0.02% (0.76)
CallEstablishRate	-0.04% (0.30)	-0.04% (0.30)	-0.04% (0.32)
VideoDuration	0.02% (0.78)	0.11% (0.31)	0.09% (0.40)
VideoFreezeDuration	-0.03% (0.76)	-2.47% (0.00)	2.43% (0.00)
VideoBitrateMean	-0.04% (0.78)	-0.54% (0.00)	-0.68% (0.00)

Table 1: Sample Experiment Scorecard - Video Decoder

Algorithm 2 Detect Loss Tolerance

```
Input: $\overrightarrow{l}$, $\overrightarrow{\delta''}$, significance level
Output: low decision impact areas
for all $(l, \delta'')$ pairs do
    Set input parameters of Algorithm 1 by $l_{trt} = l_{ctrl} = l$ and $\beta_{int} = \delta''$
    Run Algorithm 1 to generate $\frac{\delta}{se(\delta)}$
    Generate p-value $p$ using standard normal distribution
end for
Flag $\overrightarrow{l} \otimes \overrightarrow{\delta''}$ where p-values is greater than significance level as safe zone
```

3.4 Data Loss Tolerance in Experimentation

Applying Algorithm 1 to various *Skype* experiments showed how much the actual treatment effect could be different from observed scorecards. However, this difference only matters if it is large enough to impact our decision. Often, decisions informed by AB tests are binary functions with two outcomes: "roll-out" or "no roll-out". We call the data loss *tolerable* if it does not reverse our decisions. In this section, we discuss how to find the maximum tolerable data loss rate, which we refer to as the *tolerance threshold*. Note that tolerance threshold is irrelevant when correlation bias exists, since there is no tolerance for significantly different loss rates between treatment and control, as this leads to incomparable samples. The solution provided in this section is used to find the data loss threshold where treatment and control have a statistically comparable level of loss (i.e., $l = l_{ctrl} = l_{trt}$).

The decisions made by experimentation are tightly bound to the p-values of outcome metrics provided by scorecards. A "roll-out/no roll-out" decision usually translates to the p-values of desired metrics being lower/higher than significance level[1]. Leveraging the binary nature of experiment decision process, we approached this problem by comparing observed p-values with simulated p-values for complete samples (no data loss). If both are on the same side of the significance level, data loss is tolerable for that experiment. Without loss of generality, we set $\delta' = \bar{y}'_{trt} - \bar{y}'_{ctrl} = 0$. This way, the observed p-value is higher than the significance threshold and we just need to track the simulated p-values.

To find a threshold that is tolerable for most experiments in a given experimentation platform, we need to run Algorithm 1 repeatedly for multiple loss rates $\overrightarrow{l}$, and under different scenarios $[\bar{y}''_{ctrl}, s''_{ctrl}, \beta_{int}]$. The input parameters of Algorithm 1 must be adjusted to reflect the target population, the nature of data loss, and the sensitivity of experiments running on that platform. That is, to fix $[\bar{y}'_{ctrl}, s'_{ctrl}, s'_{trt}, \bar{y}'_{trt}]$ to the known summary statistics of the target population and to estimate $\bar{y}''_{ctrl}, s''_{ctrl}$ according to assumptions about the overall user experience under lossy conditions.

[1] The significance level can be different for each organization depending the number of metrics they consider for experimentation and accepted level of false positive rate. A popular significance level for pairwise comparison on single metric is 0.05.

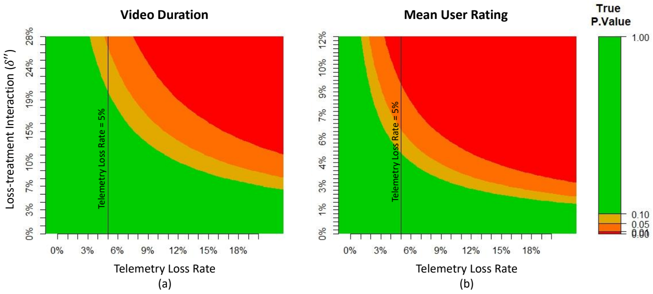


Figure 2: Loss Tolerance plot for Video Duration on a low-end mobile platform (a); and Mean User Rating on a high-end mobile platform (b).

Therefore, most input parameters in Algorithm 1 are constant values reflecting the specifications of the experimentation platform, while β_{int} varies over a possible range of values that represent different sensitivity levels for experiments run in the platform. Since δ' is set to zero, $\beta_{int} = \bar{y}''_{trt} - \bar{y}''_{ctrl} = \delta''$ according to formula (9). We use δ'' instead of β_{int} for this simulation because it is easier to interpret for engineering teams, whose input is required for defining the range of β_{int}.

With the above settings, Algorithm 1 can be used to generate a 3-dimensional array of $[\overrightarrow{l}, \overrightarrow{\delta}'', \overrightarrow{p}]$ where p is the simulated p-value for a complete sample based on given loss rate and experiment sensitivity level.

Discussion on setting input parameters: Prior knowledge about the nature of data loss plays an important role in setting these parameters. In *Skype*, we found that the causes of data loss are related to poor experience. A conservative choice is to set values drawn from the poor experience population (i.e., simulating $\bar{y}'_{ctrl}, s'_{ctrl}$ from the lower tail of outcome metric distribution). Choosing values closer to the distribution center leads to a higher tolerance compatible with the MAR assumption. It is also important to capture a realistic range for $\delta'' = \bar{y}''_{trt} - \bar{y}''_{ctrl}$ based on prior lab tests and metric noise levels. In the absence of such knowledge, $(0, 2 \times S)$ where S is metric standard deviation, is a reasonable range.

Application to *Skype* Experimentation: At *Skype*, we applied Algorithm 2 to various segments of the target population defined by device platforms (e.g., Android, Windows) due to the heterogeneity of metric noise levels and baselines across different platforms. The following provides further detail on the input parameters:

(1) The summary statistics of lost data points in control group are set to the lower 10th percentile of the respective metrics
(2) $\overrightarrow{l}$ ranges from 0 to 20% to cover all possible loss rates
(3) $\overrightarrow{\delta''}$ ranges from 0 to the minimum of 30% of metric average and 50% of its standard deviation
(4) The significance level is set to 0.01

As shown in Figure 2, higher loss rates increase the chance of decisions being reversed. However, the rate of increments varies by metric and platform. Average User Rating on a high-end mobile platform, for example, has higher sensitivity to data loss compared to Video Duration on a mobile low-end platform. This is shown as larger safe zone (green area) for data loss on Video Duration.

Algorithm 3 Data loss estimation using the anchor method.

```
procedure AnchorMethod($server\_events\_set$, $client\_events\_set$)
    $events\_lost = 0$
    $expected\_events = $ Count($server\_events\_set$)

    for all $leg\_id \in server\_events\_set$ do
        if Not(HasKey($client\_events\_set$, $leg\_id$)) then
            $events\_lost \leftarrow events\_lost + 1$
        end if
    end for

    $anchor\_loss\_rate \leftarrow \frac{events\_lost}{expected\_events}$
    return $anchor\_loss\_rate$
end procedure
```

These two examples show how variable the tolerance could be depending on experiments' sensitivity to loss. Experiments that are designed to enhance user experience in poor network conditions, for example, may fall in the top area in these plots and have lower tolerance for data loss. In practice, there is no unique non-zero tolerance threshold that uniformly satisfies all types of experiments. We set the overall target for data loss reduction to the threshold value that would ensure trustworthy experiments for majority of treatment types. According to the simulation results, 5% is the threshold value that provides tolerable coverage and risk of false positives/negatives for *Skype*.

4 MEASURING DATA LOSS

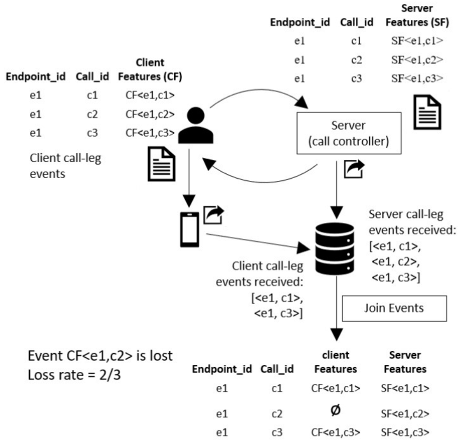
In this section, we first present two approaches to measure loss of telemetry events. Then, we evaluate these techniques in practice, using the results from various experiments. Finally, we provide some recommendations on how telemetry loss can be reduced.

4.1 Anchor Method

This method relies on pairing (or anchoring) client events with a more reliable server event. Call establishment in *Skype* is negotiated using a server. Therefore, for every call (uniquely identified by *call_id*), a record of the call's technical telemetry and the participating legs is reported by the server. Server machines are reliable, not bandwidth-constrained, and therefore exhibit near-zero telemetry loss. The unreliable client events can therefore be paired with the highly reliable server events to get an estimate of the loss in client telemetry. Each leg of the call is uniquely identified by $leg\_id = (call\_id, endpoint\_id)$

The anchor method is shown in Figure 3 and Algorithm 3. Figure 3 shows the table of events reported by the client and server, where each row corresponds to a call. Note that the figure only shows one leg of the call. For each of these call legs, the server also logs the leg IDs, allowing the client and server leg events to be paired. In this example, measures of event $CF < e1, c2 >$ could not be uploaded and this constitutes a loss.

Note that the anchor method is limited to scenarios where both client and server events are submitted. For example, loss can be measured only for call attempts where the request/acknowledgment reached the server. Therefore, this method can be applied to all *established* calls but not all *attempted* calls.



Endpoint_id	Call_id	Client Features (CF)
e1	c1	CF<e1,c1>
e1	c2	CF<e1,c2>
e1	c3	CF<e1,c3>

Endpoint_id	Call_id	Server Features (SF)
e1	c1	SF<e1,c1>
e1	c2	SF<e1,c2>
e1	c3	SF<e1,c3>

Endpoint_id	Call_id	client Features	Server Features
e1	c1	CF<e1,c1>	SF<e1,c1>
e1	c2	Ø	SF<e1,c2>
e1	c3	CF<e1,c3>	SF<e1,c3>

Figure 3: The Anchor method to estimate data loss.

4.2 Sequence Method

The sequence method (Algorithm 4) is a more general solution compared to the anchor method, using a monotonically increasing counter that is persisted to the client's local storage. This counter is referred to as a *sequence number* (*sn*). Each event has an associated counter per endpoint to track the number of events that have been generated. This monotonically increasing set is referred to as a *sequence*, as shown in Figure 4. At the beginning of each call, the *sn* value for each event is incremented and reported as part of the event. After the telemetry uploader transmits the client events, the associated *sn* values can be reconstructed. If there is no loss in events, there should be no gaps in the *sn* values when considered in sorted order; a gap in *sn* indicates that a client event has been lost, with the size of the gap indicating the number of lost events.

Algorithm 4 shows how to compute the sequence loss for a batch of de-duplicated events. In practice, in a live system, the backend system needs to maintain a lookup table of the sequence information. Specifically, the lookup table needs to maintain the following counters: 1) last received sequence number, *prev_sn*; 2) the cumulative loss so far, *sequence_gap*; and 3) the size of the sequence, *expected_sequence_size*. By accumulating the *sequence_gap* values and *expected_sequence_size* values across all sequences, we can incrementally compute the overall data loss.

The sequence method can be applied in situations where there is no reliable server method to pair with, such as randomly sampled events that do not have a reliable anchor (e.g., user ratings). As a practical matter, *sn* values are reset when users uninstall and reinstall the app (common in low-end devices as users try to conserve space by removing applications when not needed). While this is a corner case, these resets need to be detected and handled properly.

4.3 Data Loss Measurements in Practice

We use the *CST* event described in Section 2.2 to show the results of the anchor and sequence methods and discuss their practical implications. Although we cannot report absolute numbers due to confidentiality restrictions, we show appropriately scaled relative values to convey the results. Since the anchor method uses server-side events as the baseline, we compare the two methods only for calls that are established (i.e., the server has a record of the caller's attempt or callee's response). We set the *MIN_SEQUENCE_SIZE* to 5, which we will explain in further detail later. The dataset used for the analysis consisted of more than a billion established *Skype* calls over a period of several weeks. The anchor and sequence methods were implemented using Microsoft's big data analysis platform [2], which has a query language is similar to Apache Hive [20]. Our implementation is part of the production telemetry and experimentation processing pipeline.

The loss rate of the *CST* client side event as estimated by the anchor and sequence methods is shown in Figure 5a. The overall absolute difference between the two methods is less than 0.5%. Not surprisingly, the absolute loss is lower for desktop platforms than mobile platforms. From our dataset, we found that the estimated difference in loss between the two methods is less than 0.3% for desktop platforms, and about 0.7% for mobile platforms. Due to the large number of events, the confidence intervals of the estimates are very small and not shown in the figure. Note that that the Sequence method consistently estimates lower loss than the Anchor method.

The length of each sequence is a function of the number of calls made by the endpoint during the time period used for the study. In general, the distribution of the sequence size will vary per application, based on the usage characteristics. Since we have

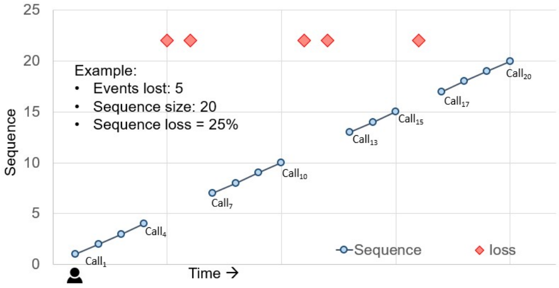


Figure 4: The Sequence method to estimate data loss.

Algorithm 4 Data loss estimation using the sequence method.

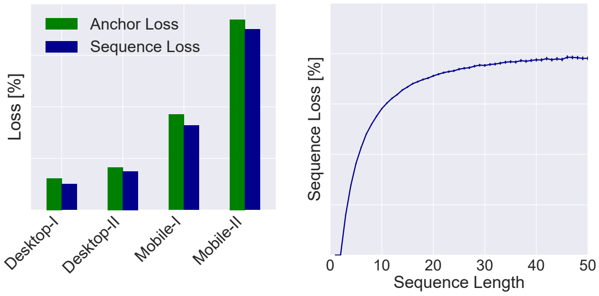
Input: $MIN\_SEQUENCE\_SIZE$

```
procedure SequenceMethod($sequence\_list$)
    $events\_lost \leftarrow 0$
    $events \leftarrow 0$
    for all $sequence \in sequence\_list$ do
        $(seq\_gap, seq\_size) \leftarrow$ SequenceLoss($sequence$)
        $events\_lost \leftarrow events\_lost + seq\_gap$
        $expected\_events \leftarrow expected\_events + seq\_size$
    end for

    $sequence\_loss\_rate \leftarrow \frac{events\_lost}{expected\_events}$
    return $sequence\_loss\_rate$
end procedure


procedure SequenceLoss($sequence$)
    $expected\_sequence\_size \leftarrow$ Max($sequence$) − Min($sequence$) + 1
    if $expected\_sequence\_size < MIN\_SEQUENCE\_SIZE$ then
        return (0, 0)
    end if

    $sequence\_gap \leftarrow expected\_sequence\_size -$ Count($sequence$)
    return ($sequence\_gap, expected\_sequence\_size$)
end procedure
```



(a)

(b)

Figure 5: (a) Estimates of data loss in the *CST* event using the Anchor and Sequence methods. (b) Impact of sequence length on sequence loss estimates for mobile platforms.

millions of users making calls at varying levels of activity, we can study the relationship between sequence size and sequence loss. Figure 5b shows this relationship for our mobile platforms, showing that the estimates converge as the size of the sequence increases. This is because the uncertainty in the loss estimate of the sequence endpoints decreases as the sequence grows. Due to this effect, the sequence method tends to underestimate the overall level of loss when sequences are short. In our experience, computing sequence loss for sequences of size greater than 5 provides a good trade-off between coverage and accuracy.

4.4 Discussion on Data Loss Estimation

The practical tradeoffs and qualitative comparisons between the sequence method and anchor method are shown in Table 2.

For the majority of the scenarios in VoIP calling, the anchor method is appropriate since we have a server side event to anchor with. Moreover, the anchor method provides better accuracy and requires less state maintenance. As a consequence, the anchor method is easier to integrate in experimentation scorecards. However, the timescale of many of the experiments conducted in *Skype* last in the order of days. At this timescale, the sequence method may not provide adequate coverage, due to a large number of sequences falling below the *MIN_SEQUENCE_SIZE* threshold. In our experiments, at least two weeks worth of data is required for sequence method to provide adequate coverage. Nevertheless, the sequence method has been used to measure loss rate of events where a good anchor is not available. Even if sequence method cannot be easily integrated in the experimentation scorecard, it can be used to establish the overall loss rate of events, and in-turn, improve the trustworthiness of the metrics derived from those events.

Table 2: Comparison of Anchor and Sequence methods

Topic	Anchor Method	Sequence Method
Dependencies	Server side event required as a source of ground truth.	Relies purely on client side telemetry.
Sampled events	Typically, a good anchor is not available.	Can be used for all events.
Accuracy	Reliable ground truth results in high accuracy.	More accurate for longer sequences.
Treatment effect bias	Loss estimates do not lead to biases between control and treatment.	Loss estimates may lead to biased comparison due to *MIN_SEQUENCE_SIZE* threshold.
State maintenance	The approach is stateless.	Table of sequence information needs to be maintained for incremental processing.
Integration with experimentation scorecard	Easier to integrate with the scorecard due to its simplicity.	Integration with scorecard is more challenging due to the state overhead.

4.5 Best Practices for Reducing Data Loss

In our experiences with *Skype*, we have successfully improved event reliability without impacting service quality using a system that is service aware, with prioritized, persistent event queues. This has helped us bring the loss to below 2% for events used in our experimentation system. While the design of the telemetry system is beyond the scope of this paper, we would like to share some of the lessons we have learned on low-loss telemetry design, including how careful design of telemetry events can make the best use of the available client resources (bandwidth, storage, etc.):

- KPI hierarchy design: Organizations need to carefully design a system of metrics with a clear hierarchy. We recommend three tiers. Tier-0 represent business health metrics, Tier-1 represents leading indicators of quality/reliability, and Tier-2 represents operational metrics of sub-components. The events should be designed and prioritized using this map. This ensures that the most informative events suffer the least loss. In an A/B test aimed at evaluating the impact of prioritization for one the business-critical KPIs, we found an absolute loss reduction of 3.8% when the priority of the event was increased by one level.
- Split large events: Keeping the size of each event small is critical to minimize congestion. Moreover, the measures in each event should provide information for metrics at the same tier. For example, mixing high priority measures/metrics (e.g., user ratings) with lower priority ones (e.g., UI selections) will cause unnecessary loss in critical information. In a lab experiment, we found that lowering the event size from 24 KB to 3 KB resulted in reducing the loss from 30% to 4%.
- Review feature importance: Over the life-cycle of a product, architectural changes and bug fixes lead to changes in the usefulness of measures. It is critical to periodically evaluate the importance of these measures by correlating them with a Tier-0 or Tier-1 metric. Measures with limited correlation should be either removed from the event or investigated to keep the size small.

While we cannot completely eliminate data loss, measuring its extent and applying the above principles to minimize it are crucial prerequisites for trustworthy experimentation.

5 PRACTICAL GUIDANCE FOR EXPERIMENTERS

Our experiences of running experiments have taught us that the impact of data loss can be very significant, and its importance can be easily underestimated. Therefore, we want to share some general guidance for the benefit of the online experimentation community. We have followed an iterative process in improving the trustworthiness of our experimentation system and its resilience to telemetry loss, as described in the steps below:

(1) Measure and track data loss for relevant events by adding them to the experimentation scorecard using the methods described in Section 4.
(2) Estimate how much loss that can be tolerated by the experimentation system using the methodology described in Section 3.3, and exclude telemetry events that suffer higher levels of loss from decision making.
(3) Use examples similar to Section 2.3 to communicate biases introduced by these losses.
(4) Mitigate the impact of loss by restructuring events using recommendations outlined in Section 4.5.
(5) Investigate the source of losses for events, starting with those with the highest loss rates.

In our experience, we found that it is critical to make the estimates of loss rates part of the experiment scorecard, reporting these alongside the other metrics monitored by the experimenter. By doing this, it becomes easier for experimenters to detect biases, and provides an extra check for the validity of experimental results. In our organization, we flag events that suffer a loss rates greater than 5%, and mark metrics relying on these events as invalid. For example, for a given experiment, suppose the loss in *CST* event was higher than 5%; in this case, the *call_duration* metric will be marked invalid for the purposes of analysis and conclusions.

We also find this 5% threshold useful when considering newly added metrics, since new events often exhibit high loss rates (10% or more). In such cases, we ask the appropriate engineering team to find the root cause and fix bugs as needed (e.g., the *ui-change* experiment discussed in Section 1), to bring the loss rate below the allowed threshold before they can be used in any experimentation analysis. Several teams have gone through this exercise. Their focused efforts have helped lower the levels of telemetry loss, and improved the overall trustworthiness of our online experiments.

6 CONCLUSION

Telemetry loss is an inseparable part of online experimentation with potentially dramatic implications. Since many apps are required to operate under resource constraints and challenging network environments, this problem is even more severe in client experiments. Based on several examples observed from real experiments at scale, we show the impact of telemetry loss on experiment outcomes. In this paper, we argue that it is vital to measure and track the level of loss in experimentation systems. We provide a theoretical framework for characterizing the types of biases introduced by telemetry loss. Using this framework, we provide a methodology for experimenters to evaluate the amount of loss that can be tolerated in their systems. To measure the absolute level of loss, we present the anchor method and sequence method: two practical approaches that have been deployed at scale. While we note that completely eliminating telemetry loss is not practical, we present the community with a set of best practices to reduce loss rates and manage the problem. These methods have already been adopted by several applications with millions of users across billions of sessions. Finally, we would like to emphasize that these principles can be applied generally, to improve the trustworthiness of any online experimentation system running at scale.

ACKNOWLEDGMENTS

We thank Sergey Sukhanov for the sequence number implementation and Sooraj Kuttykrishnan for his useful comments and feedback. We also thank the data pipeline, experimentation, and telemetry management teams within Skype/Microsoft; without their significant contributions, we would be flying blind to telemetry loss.

REFERENCES

[1] Fabio Celli et al. 2016. Predicting Brexit: Classifying agreement is better than sentiment and pollsters. In *Proc. Workshop on Computational Modeling of People's Opinions, Personality, and Emotions in Social Media.*
[2] Ronnie Chaiken et al. 2008. SCOPE: easy and efficient parallel processing of massive data sets. *Proc. VLDB Endowment* 1, 2 (2008).
[3] Alex Deng et al. 2013. Improving the Sensitivity of Online Controlled Experiments by Utilizing Pre-Experiment Data. In *Proc. Conference on Web Search and Data Mining.*
[4] Pavel Dmitriev et al. 2017. A/B Testing at Scale: Accelerating Software Innovation. In *Proc. ACM KDD '17.*
[5] Pavel Dmitriev et al. 2017. A Dirty Dozen: Twelve Common Metric Interpretation Pitfalls in Online Controlled Experiments. In *Proc. ACM KDD '18.*
[6] Aleksander Fabijan et al. 2017. The Benefits of Controlled Experimentation at Scale. In *Proc. Euromicro Conference on Software Engineering and Advanced Analytics.*
[7] Jim Gray et al. 1996. The dangers of replication and a solution. *ACM SIGMOD Record* 25, 2 (1996).
[8] Kosuke Imai. 2009. Statistical analysis of randomized experiments with non-ignorable missing binary outcomes: an application to a voting experiment. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 58, 1 (2009).
[9] Junchen Jiang et al. 2016. Via: Improving internet telephony call quality using predictive relay selection. In *Proc. ACM SIGCOMM '16.*
[10] Scott Keeter. 2006. The impact of cell phone noncoverage bias on polling in the 2004 presidential election. *Public Opinion Quarterly* 70, 1 (2006).
[11] Ron Kohavi et al. 2009. Controlled experiments on the web: survey and practical guide. In *Proc. ACM KDD '09.*
[12] Ron Kohavi et al. 2013. Seven Rules of Thumb for Web Site Experimenters. In *Proc. ACM KDD '13.*
[13] Ron Kohavi and Stefan Thomke. 2017. The surprising power of online experiments. *Harvard Business Review* 95, 5 (2017).
[14] Adam Langley et al. 2017. The QUIC transport protocol: Design and Internet-scale deployment. In *Proc. ACM SIGCOMM '17.*
[15] Roderick JA Little and Donald B Rubin. 2014. *Statistical analysis with missing data.* Vol. 333. John Wiley & Sons.
[16] Francesca Molinari. 2010. Missing Treatments. *Journal of Business and Economic Statistics* 28, 1 (2010).
[17] Douglas C. Montgomery. 2008. *Design and Analysis of Experiments.* John Wiley & Sons.
[18] H Schulzrinne et al. 2003. RTP: A Transport Protocol for Real-Time Applications. *Internet RFCs* RFC 3550 (2003).
[19] Diane Tang et al. 2010. Overlapping experiment infrastructure: More, better, faster experimentation. In *Proc. ACM KDD '10.*
[20] Ashish Thusoo et al. 2009. Hive: a warehousing solution over a map-reduce framework. *Proc. VLDB Endowment* 2, 2 (2009).
[21] Matthias Wiesmann et al. 2000. Database replication techniques: A three parameter classification. In *Proc. IEEE Symposium on Reliable Distributed Systems.*
[22] Ya Xu et al. 2015. From infrastructure to culture: A/B testing challenges in large scale social networks. In *Proc. ACM KDD '15.*
[23] Ya Xu and Nanyu Chen. 2016. Evaluating Mobile Apps with A/B and Quasi A/B Tests. In *Proc. ACM KDD '16.*