PrivApprox: Privacy-Preserving Stream Analytics

https://PrivApprox.github.io

Do Le Quoc[†], Martin Beck[†], Pramod Bhatotia[*], Ruichuan Chen[‡], Christof Fetzer[†], Thorsten Strufe[†]

[†]TU Dresden [*]University of Edinburgh [‡]Nokia Bell Labs

Abstract

How to preserve users' privacy while supporting high-utility analytics for low-latency stream processing?

To answer this question: we describe the design, implementation and evaluation of PRIVAPPROX, a data analytics system for privacy-preserving stream processing. PRIVAPPROX provides three important properties: *(i)* Privacy: zero-knowledge privacy guarantee for users, a privacy bound tighter than the state-of-the-art differential privacy; *(ii)* Utility: an interface for data analysts to systematically explore the trade-offs between the output accuracy (with error estimation) and the query execution budget; *(iii)* Latency: near real-time stream processing based on a scalable "synchronization-free" distributed architecture.

The key idea behind our approach is to marry two techniques together, namely, *sampling* (used for approximate computation) and *randomized response* (used for privacy-preserving analytics). The resulting marriage is complementary — it achieves stronger privacy guarantees, and also improves the performance for stream analytics.

1 Introduction

Many online services continuously collect users' private data for real-time analytics. Much of this data arrives as a data stream and in huge volumes, requiring real-time stream processing based on distributed systems [1–3, 21].

In the current ecosystem of data analytics, the analysts usually have direct access to users' private data, and must be trusted not to abuse it. However, this trust has been violated in the past [28, 49, 62, 69]. A pragmatic ecosystem has two desirable, but contradictory design requirements: *(i)* stronger privacy guarantees for users, and *(ii)* high-utility stream analytics in real time. Users seek stronger privacy, while analysts strive for high-utility analytics in real time.

To meet these two design requirements, there is a surge of novel computing paradigms that address these concerns, albeit *separately*. Two such paradigms are *privacy-preserving analytics* to protect user privacy and *approximate computation* for real-time analytics.

Privacy-preserving analytics. Recent privacy-preserving analytics systems favor a distributed architecture to avoid central trust (see §8 for details), where users' private data is stored locally on their respective client devices. Data analysts use a publish-subscribe mechanism to run aggregate queries over the distributed private dataset of a large number of clients. Thereafter, such systems add noise to the aggregate output to provide useful privacy guarantees, such as differential privacy [32]. Unfortunately, these state-of-the-art systems normally deal with single-shot batch queries, and therefore, these systems cannot be used for real-time stream analytics.

Approximate computation. Approximate computation is based on the observation that many data analytics jobs are amenable to an approximate rather than the exact output (see §8 for details). Such applications include speech recognition, computer vision, machine learning, and recommender systems. For such an approximate workflow, it is possible to trade accuracy by computing over a subset (usually selected via a sampling mechanism) instead of the entire input dataset. Thereby, data analytics systems based on approximate computation can achieve low latency and efficient utilization of resources. However, the existing systems for approximate computation assume a centralized dataset, where the desired sampling mechanism can be employed. Thus, existing systems are not compatible with the distributed privacy-preserving analytics systems.

The marriage. In this paper, we make the observation that the two computing paradigms, i.e., privacy-preserving analytics and approximate computation, are complementary. Both paradigms strive for an approximate instead of the exact output, but they differ in their *means* and *goals* for approximation. Privacy-preserving analytics adds explicit *noise* to the aggregate query output to protect user privacy, whereas approximate computation relies on a representative *sampling* of the entire dataset to compute over only a subset of data items to enable low-latency/efficient analytics. Therefore, we marry these two existing paradigms together in order to leverage the benefits of both. The high-level

idea is to achieve privacy (via approximation) by directly computing over a subset of sampled data items (instead of computing over the entire dataset) and then adding an explicit noise for privacy preservation.

To realize this marriage, we designed an approximation mechanism that also achieves privacy-preserving goals for stream analytics. Our design (see Figure 1) targets a distributed setting, similar as aforementioned, where users' private data is stored locally on their respective personal devices, and an analyst issues a streaming query for analytics over the distributed private dataset of users. The analyst's streaming query is executed on the users' data periodically (a configurable epoch) and the query results are transmitted to a centralized aggregator via a set of proxies. The analyst interfaces with the aggregator to get the aggregate query output periodically.

We employ two core techniques to achieve our goal. Firstly, we employ *sampling* [60] directly at the user site for approximate computation, where each user randomly decides whether to participate in answering the query in the current epoch. Since we employ sampling at the data source, instead of sampling at a centralized infrastructure, we are able to squeeze out the desired data size (by controlling the sampling parameter) from the very first stage in the analytics pipeline, which is essential in low-latency environments.

Secondly, if the user participates in the query answering process, we employ a *randomized response* [37] mechanism to add noise to the query output at the user site, again locally at the source of the data in a decentralized fashion. In particular, each user locally randomizes the truthful answer to the query to achieve the differential privacy guarantees (§3.2.2). Since we employ noise addition at the source of data, instead of adding the explicit noise to the aggregate output at a trusted aggregator or proxies, we enable a truly "synchronization-free" distributed architecture, which requires *no coordination* among proxies and the aggregator for the mandated noise addition.

The last, but not the least, silver bullet of our design: it turns out that the combination of the two aforementioned techniques (i.e., sampling and randomized response) leads us to achieve zero-knowledge privacy [41], a privacy bound tighter than the state-of-the-art differential privacy [32].

To summarize, we present the design and implementation of a practical system for privacy-preserving stream analytics in real time. In particular, our system is a novel combination of the sampling and randomized response techniques, as well as a scalable "synchronization-free" routing scheme which employs a light-weight XOR-based encryption scheme [26]. The resulting system ensures zero-knowledge privacy, anonymization, and unlinkability for users (§2.2). Altogether, we make the following contributions:

- We present a marriage of the sampling and randomized response techniques to achieve improved performance and stronger privacy guarantees.

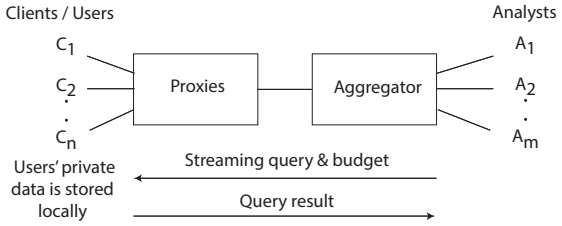

Figure 1: System overview.

- We present an adaptive query execution interface for analysts to systematically make a trade-off between the output accuracy and the query execution budget.
- We present a confidence metric on the output accuracy using a confidence interval to interpret the approximation due to sampling and randomization.

To empirically evaluate our approach, we implemented our design as a fully functional prototype in a system called PRIVAPPROX based on Apache Flink [21] and Apache Kafka [7]. In addition to stream analytics, we further extended our system to support privacy-preserving "historical" batch analytics over users' private datasets. The evaluation based on micro-benchmarks and real-world case studies shows that this marriage is, in fact, made in heaven!

2 Overview

2.1 System Architecture

PRIVAPPROX is designed for privacy-preserving stream analytics on distributed users' private dataset. Figure 1 depicts the high-level architecture of PRIVAPPROX. Our system consists of four main components: clients, proxies, aggregator, and analysts.

Clients locally store users' private data on their respective personal devices, and subscribe to queries from the system. *Analysts* publish streaming queries to the system, and also specify a query execution budget. The query execution budget can either be in the form of latency guarantees/SLAs, output quality/accuracy, or the computing resources for query processing. Our system ensures that the computation remains within the specified budget.

At a high-level, the system works as follows: a query published by an analyst is distributed to clients via the aggregator and proxies. Clients answer the analyst's query locally over the users' private data using a privacy-preserving mechanism. Client answers are transmitted to the aggregator via anonymizing *proxies*. The *aggregator* aggregates received answers from the clients to provide privacy-preserving stream analytics to the analyst.

2.2 System Model

Query model. PRIVAPPROX supports the SQL query language for *analysts* to formulate streaming queries, which are executed periodically at the clients as sliding window computations [14]. While queries can be complex, the results of a query are expressed as counts within histogram buckets, i.e.,

each bucket represents a range of the query's answer values. Specifically, each query answer is represented in the form of binary buckets, where each bucket stores a value '1' or '0' depending on whether or not the answer falls into the value range represented by that bucket. For example, an analyst can learn the driving speed distribution across all vehicles in San Francisco by formulating an SQL query "`SELECT speed FROM vehicle WHERE location='San Francisco'`". The analyst can then define 12 answer buckets on speed: '0', '1∼10', '11∼20', ···, '81∼90', '91∼100', and '>100'. If a vehicle is moving at 15 mph in San Francisco, it answers '1' for the third bucket and '0' for all others.

Our query model supports not only numeric queries as described above, but also non-numeric queries. For non-numeric queries, each bucket is specified by a matching rule or a regular expression. Note that, at first glance, our query model may appear simple; however, it has been shown to be effective for a wide-range of analytics algorithms [19, 20].

Threat model. *Analysts* are potentially malicious. They may try to violate the PRIVAPPROX's privacy model (described later), i.e., de-anonymize clients, build profiles through the linkage of queries and answers, or remove the added noise from answers.

Clients are potentially malicious. They could generate false or invalid responses to distort the query result for the analyst. However, we do not defend against the Sybil attack [31], which is beyond the scope of this work [75].

Proxies are also potentially malicious. They may transmit messages between clients and the aggregator in contravention of our system protocols. PRIVAPPROX includes at least two proxies, and there are at least two proxies which do not collude with each other.

The *aggregator* is assumed to be honest-but-curious. The aggregator faithfully conforms to the system protocols, but may try to exploit the information about clients. The aggregator does not collude with any proxy nor the analyst.

Finally, we assume that all the end-to-end communications use authenticated and confidential connections (e.g., protected by long-lived TLS connections), and no system component could monitor all network traffic.

Privacy model. Our privacy properties include: *(i)* zero-knowledge privacy, *(ii)* anonymity, and *(iii)* unlinkability.

All aggregate query results in the system are independently produced under the *zero-knowledge privacy* guarantees [41]. The zero-knowledge privacy metric builds upon differential privacy [32], and provides a tighter bound on privacy guarantees compared to differential privacy. Informally, zero-knowledge privacy states that essentially everything that an adversary can learn from the output of an zero-knowledge private mechanism could also be learned using the aggregate information. *Anonymity* means that no system component can associate query answers or query requests with a specific client. Finally, *unlinkability* means that no system component can join any pair of query requests or answers to the same client, even to the same anonymous client.

We give a sketch of the privacy analysis in §4, while we also provide the formal definition, analysis, and proof in the technical report [64].

3 Design

PRIVAPPROX consists of two main phases (see Figure 1): *submitting queries* and *answering queries*. In the first phase, an analyst submits a query (along with the execution budget) to clients via the aggregator and proxies. In the second phase, the query is answered by the clients in the reverse direction.

3.1 Submitting Queries

To perform statistical analysis over users' private data streams, an analyst creates a query using the query model described in §2.2. In particular, each query consists of the following fields, and is signed by the analyst for non-repudiation:

$$Query := \langle Q_{ID}, SQL, A[n], f, w, \delta \rangle \quad (1)$$

- Q_{ID} denotes a unique identifier of the query. This can be generated by concatenating the identifier of the analyst with a serial number unique to the analyst.
- *SQL* denotes the actual SQL query, which is passed on to clients and executed on their respective personal data.
- $A[n]$ denotes the format of a client's answer to the query. The answer is an n-bit vector where each bit associates with a possible answer value in the form of a "0" or "1" per index (or answer value range).
- f denotes the answer frequency, i.e., how often the query needs to be executed at clients.
- w denotes the window length for sliding window computations [13]. For example, an analyst may only want to aggregate query results for the last ten minutes, which means the window length is ten minutes.
- δ denotes the sliding interval for sliding window computations. For example, an analyst may want to update the query results every one minute, and so the sliding interval is set to one minute.

After forming the query, the analyst sends the query, along with the query execution budget, to the aggregator. Once receiving the pair of the query and query budget from the analyst, the aggregator first converts the query budget into system parameters for sampling (s) and randomization (p,q). We explain these system parameters in the next section §3.2. Hereafter, the aggregator forwards the query and the converted system parameters to clients via proxies.

3.2 Answering Queries

After receiving the query and system parameters, we next explain how the query is answered by clients and processed by the system to produce the result for the analyst. The query answering process involves four steps including *(i)* sampling at clients for low-latency approximation; *(ii)* randomizing answers for privacy preservation; *(iii)* transmitting answers

via proxies for anonymization and unlinkability; and finally, *(iv)* aggregating answers with error estimation to give a confidence level on the approximate result.

3.2.1 Step I: Sampling at Clients

We make use of approximate computation to achieve low-latency execution by computing over a subset of data items instead of the entire input dataset. Specifically, our work builds on sampling-based techniques [8, 9, 42, 53, 65] in the context of "Big Data" analytics. Since we aim to keep the private data stored at individual clients, PRIVAPPROX applies an input data sampling mechanism locally at the clients. In particular, we use *Simple Random Sampling* (SRS) [60].

Simple Random Sampling (SRS). SRS is considered as a fair way of selecting a sample from a given population since each individual in the population has the same chance of being included in the sample. We make use of SRS at the clients to select clients that will participate in the query answering process. In particular, the aggregator passes the *sampling parameter* (s) on to clients as the probability of participating in the query answering process. Thereafter, each client flips a coin with the probability based on the sampling parameter (s), and decides whether to participate in answering a query. Suppose that we have a population of U clients, and each client i has an answer a_i. We want to calculate the sum of these answers across the population, i.e., $\sum_{i=1}^{U} a_i$. To compute an approximate sum, we apply the SRS at clients to get a sample of U' clients. The estimated sum is then calculated as follows:

$$\hat{\tau} = \frac{U}{U'} \sum_{i=1}^{U'} a_i \pm error \tag{2}$$

Where the error bound *error* is defined as:

$$error = t\sqrt{\widehat{Var}(\hat{\tau})} \tag{3}$$

Here, t is a value of the t-distribution with $U'-1$ degrees of freedom at the $1-\alpha/2$ level of significance, and the estimated variance $\widehat{Var}(\hat{\tau})$ of the sum is:

$$\widehat{Var}(\hat{\tau}) = \frac{U^2}{U'}\sigma^2\left(\frac{U-U'}{U}\right) \tag{4}$$

Where σ^2 is the sample variance of the sum.

Note that, in this paper, we assume that all clients produce the input stream with data items following the same distribution, i.e., all clients' data streams belong to the same stratum. We further extend our sampling mechanism with the *stratified sampling* technique [53] to deal with varying distributions of data streams. We cover the algorithm and evaluation of stratified sampling in the technical report [64].

3.2.2 Step II: Answering Queries at Clients

Clients that participate in the query answering process make use of the *randomized response* technique [37] to preserve answer privacy, with *no* synchronization among clients.

Randomized response. Randomized response protects user's privacy by allowing individuals to answer sensitive queries without providing truthful answers all the time, yet it allows analysts to collect statistical results. Randomized response works as follows: suppose an analyst sends a query to individuals to obtain the statistical result about a sensitive property. To answer the query, a client locally randomizes its answer to the query [37]. Specifically, the client flips a coin, if it comes up heads, then the client responds its truthful answer; otherwise, the client flips a second coin and responds "Yes" if it comes up heads or "No" if it comes up tails. The privacy is preserved via the ability to refuse responding truthful answers.

Suppose that the probabilities of the first coin and the second coin coming up heads are p and q, respectively. The analyst receives N randomized answers from individuals, among which R_y answers are "Yes". Then, the number of original truthful "Yes" answers before the randomization process can be estimated as:

$$E_y = \frac{R_y - (1-p)\times q \times N}{p} \tag{5}$$

Suppose A_y and E_y are the actual and the estimated numbers of the original truthful "Yes" answers, respectively. The accuracy loss η is then defined as:

$$\eta = \left|\frac{A_y - E_y}{A_y}\right| \tag{6}$$

It has been proven in [36] that, the randomized response mechanism achieves ε-differential privacy [32], where:

$$\varepsilon = \ln\left(\frac{\Pr[Response = Yes | Truth = Yes]}{\Pr[Response = Yes | Truth = No]}\right) \tag{7}$$

More specifically, the above randomized response mechanism achieves ε-differential privacy, where:

$$\varepsilon = \ln\left(\frac{p + (1-p)\times q}{(1-p)\times q}\right) \tag{8}$$

The reason is that, if a truthful answer is "Yes", then with the probability of '$p + (1-p)\times q$', the randomized answer will still remain "Yes". Otherwise, if a truthful answer is "No", then with the probability of '$(1-p)\times q$', the randomized answer will become "Yes".

It is worth mentioning that, combining the randomized response with the sampling technique described in Step I, we achieve not only differential privacy but also zero-knowledge privacy [41] which is a privacy bound tighter than differential privacy. We sketch out the proof in §4, with details in the technical report [64].

3.2.3 Step III: Transmitting Answers via Proxies

After producing randomized responses, clients transmit them to the aggregator via the proxies. To achieve anonymity

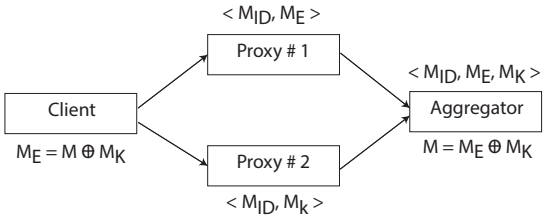

Figure 2: XOR-based encryption with two proxies.

and unlinkability of the clients against the aggregator and analysts, we utilize the XOR-based encryption together with source rewriting, which has been used for anonymous communications [26, 27, 30, 67].

XOR-based encryption. At a high-level, the XOR-based encryption employs extremely efficient bit-wise XOR operations as its cryptographic primitive compared to expensive public-key cryptography. This allows us to support resource-constrained clients, e.g., smartphones and sensors. The underlying idea of this encryption is simple: if Alice wants to send a message M of length l to Bob, then Alice and Bob share a secret M_K (in the form of a random bit-string of length l). To transmit the message M privately, Alice sends an encrypted message '$M_E = M \oplus M_K$' to Bob, where '$\oplus$' denotes the bit-wise XOR operation. To decrypt the message, Bob again uses the bit-wise XOR operation: $M = M_E \oplus M_K$.

Specifically, we apply the XOR-based encryption to transmit clients' randomized answers as follows. At first, each randomized answer is concatenated with its associated query identifier Q_{ID} to build a message M:

$$M = Q_{ID}, RandomizedAnswer \tag{9}$$

Thereafter, the client generates $(n-1)$ random l-bit key strings M_{K_i} with $2 \leq i \leq n$ using a cryptographic pseudo-random number generator (PRNG) seeded with a cryptographically strong random number. The XOR of all $(n-1)$ key strings together forms the secret M_K.

$$M_K = \bigoplus_{i=2}^{n} M_{K_i} \tag{10}$$

Next, the client performs an XOR operation with M and M_K to produce an encrypted message M_E.

$$M_E = M \oplus M_K \tag{11}$$

As a result, the message M is split into n messages $\langle M_E, M_{K_2}, \cdots, M_{K_n} \rangle$. Afterwards, a unique message identifier M_{ID} is generated, and sent along with the split messages to the n proxies via anonymous channels enabled by source rewriting [30, 67].

$$\begin{aligned} &\text{Client} \longrightarrow \text{Proxy1} : \langle M_{ID}, M_E \rangle \\ &\text{Client} \longrightarrow \text{Proxy}i : \langle M_{ID}, M_{K_i} \rangle \end{aligned} \tag{12}$$

Upon receiving the messages (either $\langle M_{ID}, M_E \rangle$ or $\langle M_{ID}, M_{K_i} \rangle$) from clients, the n proxies transmit these messages to the aggregator.

The message identifier M_{ID} ensures that M_E and all associated M_{K_i} will be joined later to decrypt the original message M at the aggregator. Note that, $\langle M_{ID}, M_E \rangle$ and all $\langle M_{ID}, M_{K_i} \rangle$ are computationally indistinguishable, which hides from the proxies if the received data contains the encrypted answer or is just a pseudo-random bit string.

3.2.4 Step IV: Generating Result at the Aggregator

At the aggregator, all data streams ($\langle M_{ID}, M_E \rangle$ and $\langle M_{ID}, M_{K_i} \rangle$) are received, and can be joined together to obtain a unified data stream. Specifically, the associated M_E and M_{K_i} are paired by using the message identifier M_{ID}. To decrypt the original randomized message M from the client, the XOR operation is performed over M_E and M_K: $M = M_E \oplus M_K$ with M_K being the XOR of all M_{K_i}: $M_K = \bigoplus_{i=2}^{n} M_{K_i}$. As the aggregator cannot identify which of the received messages is M_E, it just XORs all the n received messages to decrypt M.

The joined answer stream is processed to produce the query results as a sliding window. For each window, the aggregator first adapts the computation window to the current start time t by removing all old data items, with $timestamp < t$, from the window. Next, the aggregator adds the newly incoming data items into the window. Then, the answers in the window are decoded and aggregated to produce the query results for the analyst. Each query result is an estimated result which is bound to a range of error due to the approximation. The aggregator estimates this error bound using equation 3 and produces a confidence interval for the result as: $queryResult \pm errorBound$. The entire process is repeated for every window.

Note that an adversarial client might answer a query many times in an attempt to distort the query result. However, we can handle this problem, for example, by applying the *triple splitting* technique [26].

Error bound estimation. We provide an error bound estimation for the aggregate query results. The accuracy loss in PRIVAPPROX is caused by two processes: *(i)* sampling and *(ii)* randomized response. Since the accuracy loss of these two processes is statistically independent (see §6), we estimate the accuracy loss of each process separately. Furthermore, Equation 2 indicates that the error induced by sampling can be described as an additive component of the estimated sum. The error induced by randomized response is contained in the a_i values in Equation 2. Therefore, independent of the error induced by randomized response, the error coming from sampling is simply being added upon. Following this, we sum up both independently estimated errors to provide the total error bound of the query results.

To estimate the accuracy loss of the randomized response process, we make use of an experimental method. We

run several micro-benchmarks at the beginning of the query answering process (without performing the sampling process) to estimate the accuracy loss caused by randomized response. We measure the accuracy loss using Equation 6.

On the other hand, to estimate the accuracy loss of the sampling process, we apply the statistical theory of the sampling techniques. In particular, we first identify a desired confidence level, e.g., 95%. Then, we compute the margin of error using Equation 3. Note that, to use this equation the sampling distribution must be nearly normal. According to the Central Limit Theorem (CLT), when the sample size U' is large enough (e.g., ≥ 30), the sampling distribution of a statistic becomes close to the normal distribution, regardless of the underlying distribution of values in the dataset [72].

3.3 Practical Considerations

We next present two design enhancements to further improve the practicality of PRIVAPPROX.

3.3.1 Historical Analytics

In addition to providing real-time data analytics, we further extend PRIVAPPROX to support historical analytics. The historical analytics workflow is essential for the data warehousing setting, where analysts wish to analyze user behaviors over a longer time period. To facilitate historical analytics, we support the batch analytics over user data at the aggregator. The analyst can analyze users' responses stored in a fault-tolerant distributed storage (e.g., HDFS) at the aggregator to get the aggregate query result over the desired time period.

We also extend the adaptive execution interface for historical analytics, where the analyst can specify query execution budget, for example, to suit dynamic pricing in spot markets in the cloud deployment. Based on the query budget, we can perform an additional round of sampling at the aggregator to ensure that the batch analytics computation remains within the query budget (see the evaluation details in the technical report [64]).

3.3.2 Query Inversion

In the current setting, some queries may result in very few truthful "Yes" answers in users' responses. For such cases, PRIVAPPROX can only achieve lower utility of query results because the fraction of truthful "Yes" answers is distant from the second randomization parameter q (see experimental results in §6). For instance, if q is set to a high value (e.g., $q=0.9$), having only a few truthful "Yes" answers will affect the overall utility of the query result. To address this issue, we propose a *query inversion* mechanism. If the fraction of truthful "Yes" answers is too small or too large compared to the q value, then the analysts can invert the query to calculate the truthful "No" answers instead of the truthful "Yes" answers. In this way, the fraction of truthful "No" answers gets closer to q, resulting in a higher utility of the query result.

4 Privacy Analysis

PRIVAPPROX achieves the strong privacy properties *(i)* differential privacy and *(ii)* zero-knowledge privacy as introduced in §2.2. This section only provides a sketch of the full proof. The detailed proof along with the empirical evaluation is available in the technical report [64].

The basic idea is that all data from the clients is already differentially private due to the use of randomized response. Furthermore, the combination with sampling at the clients makes it zero-knowledge private as well. Following the privacy definitions [32, 41], any computation upon the results of differentially as well as zero-knowledge private algorithms is guaranteed to be private.

Intuitively, differential privacy limits the information that can be learned about any individual i by the difference occurring from either including i's sensitive data in a differentially private computation or not. Zero-knowledge privacy on the other hand also gives the adversary access to aggregate information about the remaining individuals. Essentially, everything that can be learned about individual i can also be learned by having access to some aggregate information upon them.

***(i)* Differential privacy.** Differential privacy is already fulfilled by randomized response [36]. However, due to the use of client-side sampling, a tighter privacy bound can be derived. Consequently, we show that sampling and randomize response commute and how to derive the combined bound given the sampling and randomize response parameters. The commutative property is shown by showing statistical indistinguishability of applying a sampling and randomize response in that order and vice versa. Furthermore, we show that sampling can be decomposed into pre- and post-sampling by leveraging the commutative property of multiplication.

***(ii)* Zero-knowledge privacy.** The zero-knowledge privacy property follows from combining a differentially private algorithm (randomized response) with an aggregation function (sampling) as given in the seminal work on zero-knowledge privacy [40]. More detail is available in the technical report [64].

5 Implementation

We implemented PRIVAPPROX as an end-to-end stream analytics system. Figure 3 presents the architecture of our prototype. Our system implementation consists of three main components: *(i)* clients, *(ii)* proxies, and *(iii)* the aggregator.

First, the query and the execution budget specified by the analyst are processed by the `initializer` module to decide on the sampling parameter (s) and the randomization parameters (p and q). These parameters along with the query are then sent to the clients.

Clients. We implemented Java-based clients for mobile devices as well as for personal computers. A client makes use of the sampling parameter (based on the `sampling` module) to decide whether to participate in the query answering process (§3.2.1). If the client decides to participate then

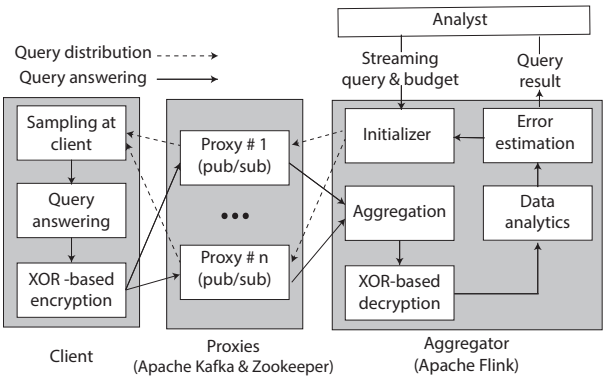

Figure 3: PRIVAPPROX architecture. Historical analytics pipeline at the aggregator is not shown for clarity.

the `query answer` module is used to execute the input query on the local user's private data stored in `SQLite` [5]. The client makes use of the randomized response to execute the query (§3.2.2). Finally, the randomized answer is encrypted using the `XOR-based encryption` module; thereafter, the encrypted message and the key messages are sent to the aggregator via proxies (§3.2.3).

Proxies. We implemented proxies based on Apache Kafka (which internally uses Apache Zookeeper [4] for fault tolerance). In Kafka, a *topic* is used to define a stream of data items. A stream *producer* can publish data items to a topic, and these data items are stored in Kafka servers called *brokers*. Thereafter, a *consumer* can subscribe to the topic and consume the data items by pulling them from the brokers. In particular, we make use of Kafka APIs to create two main topics: *key* and *answer* for transmitting the key message stream and the encrypted answer stream in the XOR-based encryption protocol, respectively (§3.2.3).

Aggregator. We implemented the aggregator using Apache Flink for real-time stream analytics and also for historical batch analytics. At the aggregator, we first make use of the join method (using the `aggregation` module) to combine the two data streams: *(i)* encrypted answer stream and *(ii)* key stream. Thereafter, the combined message stream is decoded (using the `XOR-based decryption` module) to reproduce the randomized query answers. These answers are then forwarded to the analytics module. The `analytics` module processes the answers to provide the query result to the analyst. Moreover, the `error estimation module` is used to estimate the error (§3.2.4), which we implemented using the Apache Common Math library. If the error exceeds the error bound target, a feedback mechanism is activated to re-tune the sampling and randomization parameters to provide higher utility in the subsequent epochs.

6 Evaluation: Microbenchmarks

We first evaluate PRIVAPPROX using microbenchmarks.

#I: Effect of sampling and randomization parameters. We measure the effect of randomization parameters on the utility and the privacy guarantee of the query results. In particular, the utility is measured by the query results' accuracy loss (Equation 6), and privacy is measured by the level of achieved zero-knowledge privacy (Equation 19 in the technical report [64]). In the experiment, we randomly generated 10,000 original answers, 60% of which are "Yes" answers. The sampling parameter s is set to 0.6.

Table 1: Utility and privacy of query results with different randomization parameters p and q.

p	q	Accuracy loss (η)	Privacy Level (ε)
0.3	0.3	0.0278	1.7047
	0.6	0.0262	1.3862
	0.9	0.0268	1.2527
0.6	0.3	0.0141	2.5649
	0.6	0.0128	2.0476
	0.9	0.0136	1.7917
0.9	0.3	0.0098	4.1820
	0.6	0.0079	3.5263
	0.9	0.0102	3.1570

Table 1 shows that different settings of the two randomization parameters, p and q, do affect the utility and the privacy guarantee of the query results. The higher p means the higher probability that a client responds with its truthful answer. As expected, this leads to higher utility (i.e., smaller accuracy loss η) but weaker privacy guarantee (i.e., higher privacy level ε). In addition, Table 1 also shows that the closer we set the probability q to the fraction of truthful "Yes" answers (i.e., 60% in this microbenchmark), the higher utility the query result provides. Nevertheless, to meet the utility and privacy requirements in various scenarios, we should carefully choose the appropriate p and q. In practice, the selection of the ε value depends on real-world applications [54].

We also measured the effect of sampling parameter on the accuracy loss. Figure 4 (a) shows that the accuracy loss decreases with the increase of sampling fraction, regardless of the settings of randomization parameters p and q. The benefits reach diminishing returns after the sampling fraction of 80%. The system operator can set the sampling fraction using resource prediction model [78–80] for any given SLA.

#II: Error estimation. To analyze the accuracy loss, we first measured the accuracy loss caused by sampling and randomized response *separately*. For comparison, we also computed the total accuracy loss after running the two processes in succession as in PRIVAPPROX. In this experiment, we set the number of original answers to 10,000 with 60% of which being "Yes" answers. We measured the accuracy loss of the randomized response process by setting the sampling parameter to 100% ($s=1$) and the randomization parameters p and q to 0.3 and 0.6, respectively. Meanwhile, we measured the accuracy loss of the sampling process without the randomized response process by setting p to 1.

Figure 4 (b) indicates that the accuracy loss caused by the two processes is statistically independent of each other. In addition, the accuracy loss of the two processes can effectively be added together to calculate the total accuracy loss.

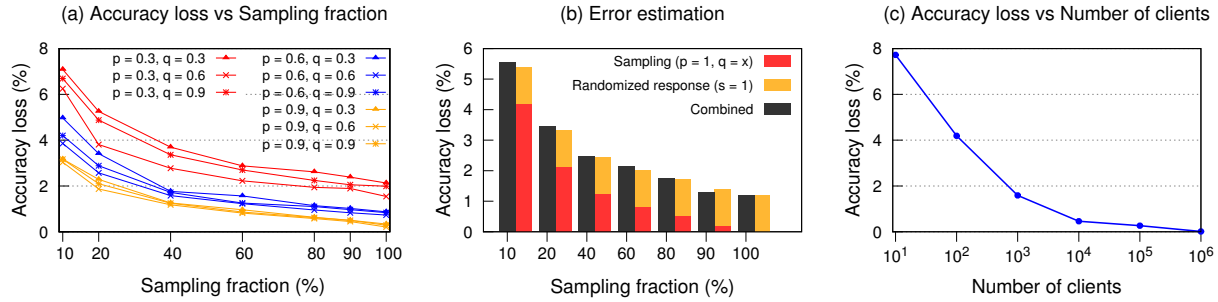

Figure 4: (a) Accuracy loss with varying sampling and randomization parameters. **(b)** Accuracy loss caused by sampling and randomized response processes, combined and individually. **(c)** Accuracy loss with varying numbers of clients.

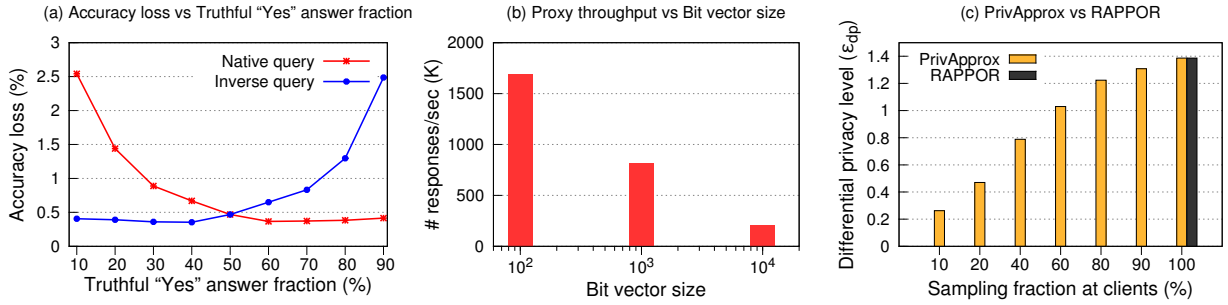

Figure 5: (a) Accuracy loss of the native and inversed query results with different fractions of truthful "Yes" answers. **(b)** Throughput at proxies with different sizes of the query answer. **(c)** Comparison between PRIVAPPROX and RAPPOR.

Table 2: Comparison of crypto overheads (# operations/sec). The public-key crypto schemes use a 1024-bit key.

	Encryption						Decryption					
	Phone		Laptop		Server		Phone		Laptop		Server	
RSA [10]	937	16×	2,770	341×	4,909	275×	126	25890×	698	23666×	859	26401×
Goldwasser [27]	2,106	7×	17,064	55×	22,902	59×	127	25686×	6,329	2610×	7,068	3209×
Paillier [66]	116	129×	489	1930×	579	2335×	72	45308×	250	66076×	309	73392×
PRIVAPPROX	15,026		943,902		1,351,937		3,262,186		16,519,076		22,678,285	

#III: Effect of the number of clients. We next analyzed how the number of participating clients affects the utility of the results. In this experiment, we fix the sampling and randomization parameters s, p and q to 0.9, 0.9 and 0.6, respectively, and set the fraction of truthful "Yes" answers to 60%.

Figure 4 (c) shows that the utility of query results improves with the increase of the number of participating clients, and few clients (e.g., <100) may lead to low-utility query results.

Note that increasing the number of participating clients leads to higher network overheads. However, we can tune the number of clients using the sampling parameter s and thus decrease the network overhead (see §7.2 #II).

#IV: Effect of the fraction of truthful answers. We measured the utility of both the native and the inversed query results with different fractions of truthful "Yes" answers. In this experiment, we still keep the sampling and randomization parameters s, p and q to 0.9, 0.9 and 0.6, respectively, and set the total number of answers to 10,000.

Figure 5 (a) shows that PRIVAPPROX achieves higher utility as the fraction of truthful "Yes" answers gets closer to 60% (i.e., the q value). In addition, when the fraction of truthful "Yes" answers y is too small compared to the q value (e.g., $y=0.1$), the accuracy loss is quite high at 2.54%. However, by using the query inversion mechanism (§3.3.2), we can significantly reduce the accuracy loss to 0.4%.

#V: Effect of answer's bit-vector sizes. We measured the throughput at proxies with various bit-vector sizes of client answers (i.e., $A[n]$ in §3.1). We conducted this experiment with a 3-node cluster (see §7.1 for the experimental setup). Figure 5 (b) shows that the throughput, as expected, is inversely proportional to the answer's bit-vector sizes.

#VI: Computational overhead of crypto operations. We compared the computational overhead of crypto operations used in PRIVAPPROX and prior systems. In particular, these crypto operations are XOR in PRIVAPPROX, RSA in [10], Goldwasser-Micali in [27], and Paillier in [66]. In this experiment, we measured the number of crypto operations that can be executed on: *(i)* Android Galaxy mini III smartphone

Table 3: Throughput (# operations/sec) at clients.

# operations/sec	Phone	Laptop	Server
SQLite read	1,162	19,646	23,418
Randomized response	168,938	418,668	1,809,662
XOR encryption	15,026	943,902	1,351,937
Total	1,116	17,236	22,026

running Android 4.1.2 with a 1.5 GHz CPU; *(ii)* MacBook Air laptop with a 2.2 GHz Intel Core i7 CPU running OS X Yosemite 10.10.2; and *(iii)* Linux server running Linux 3.15.0 equipped with a 2.2 GHz CPU with 32 cores.

Table 2 shows that the XOR operation is extremely efficient compared with the other crypto mechanisms. This highlights the importance of XOR encryption in our design.

#VII: Throughput at clients. We measured the throughput at clients. In particular, we measured the number of operations per second that can be executed at clients for the query answering process. In this experiment, we used the same set of devices as in the previous experiment. Table 3 presents the throughput at clients. To closely investigate the overheads, we measured the individual throughput of three sub-processes in the query answering process: *(i)* database read, *(ii)* randomized response, and *(iii)* XOR encryption. The result indicates that the performance bottleneck in the answering process is actually the database read operation.

#VIII: Comparison with related work. First, we compared PRIVAPPROX with SplitX [26], a high-performance privacy-preserving analytics system. Since PRIVAPPROX and SplitX share the same architecture, we compare the latency incurred at proxies in both systems.

Figure 6 shows that, with different numbers of clients, the latency incurred at proxies in PRIVAPPROX is always nearly one order of magnitude lower than that in SplitX. The reason is simple: unlike PRIVAPPROX, SplitX requires synchronization among its proxies to process query answers in a privacy-preserving fashion. This synchronization creates a significant delay in processing query answers, making SplitX unsuitable for dealing with large-scale stream analytics. More specifically, in SplitX, the processing at proxies consists of a few sub-processes including adding noise to answers, answer transmission, answer intersection, and answer shuffling; whereas, in PRIVAPPROX, the processing at proxies contains only the answer transmission. Figure 6 also shows that with 10^6 clients, the latency at SplitX is 40.27 sec, whereas PRIVAPPROX achieves a latency of just 6.21 sec, resulting in a 6.48× speedup compared with SplitX.

Next, we compared PRIVAPPROX with a recent privacy-preserving analytics system called RAPPOR [73]. Similar to PRIVAPPROX, RAPPOR applies a randomized response mechanism to achieve differential privacy. However, RAPPOR is not designed for stream analytics, and therefore, we compared PRIVAPPROX with RAPPOR for privacy only. To make an "apples-to-apples" comparison between PRIVAPPROX and RAPPOR in terms of privacy, we make a mapping

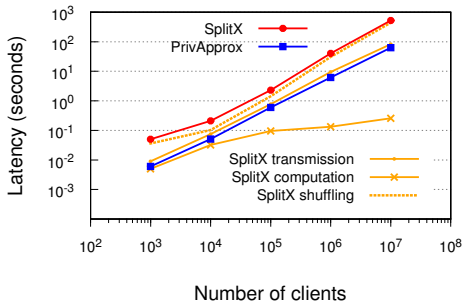

Figure 6: Comparison between SplitX and PRIVAPPROX.

between the system parameters of the two systems. We set the sampling parameter $s=1$, and the randomized parameters $p=1-f$, $q=0.5$ in PRIVAPPROX, where f is the parameter used in the randomized response process of RAPPOR [73]. In addition, we set the number of hash functions used in RAPPOR to 1 ($h=1$) for a fair comparison. In doing so, the two systems have the same randomized response process. However, since PRIVAPPROX makes use of the sampling mechanism before performing the randomized response, PRIVAPPROX achieves stronger privacy. Figure 5 (c) shows the differential privacy level of RAPPOR and PRIVAPPROX with different sampling fractions s. It is worth mentioning that, by applying the sampling mechanism, PRIVAPPROX achieves stronger privacy (i.e., zero-knowledge privacy) for clients.

7 Evaluation: Case Studies

We next present our experience of using PRIVAPPROX in the following two case studies: *(i)* New York City (NYC) taxi ride, and *(ii)* household electricity consumption.

7.1 Experimental Setup

Cluster setup. We used a cluster of 44 nodes connected via a Gigabit Ethernet. Each node contains 2 Intel Xeon quad-core CPUs and 8 GB of RAM running Debian 5.0. We deployed two proxies with Apache Kafka, each of which consists of 4 Kafka broker nodes and 3 Zookeeper nodes. We used 20 nodes to deploy Apache Flink as the aggregator. In addition, we employed the remaining 10 nodes to replay the datasets to generate data streams for the evaluation.

Datasets. For the first case study, we used the *NYC Taxi Ride* dataset from the DEBS 2015 Grand Challenge [51]. For the second case study, we used the *Household Electricity Consumption* dataset [6].

Queries. For the NYC taxi ride case study, we created a query: "*What is the distance distribution of taxi rides in New York?*". We defined the query answer with 11 buckets as follows: [0, 1) mile, [1, 2) miles, [2, 3) miles, [3, 4) miles, [4, 5) miles, [5, 6) miles, [6, 7) miles, [7, 8) miles, [8, 9) miles, [9, 10) miles, and [10, $+\infty$) miles.

For the second case study, we defined a query to analyze the electricity usage distribution of households over the past 30 minutes. The query answer format is as follows: [0, 0.5]

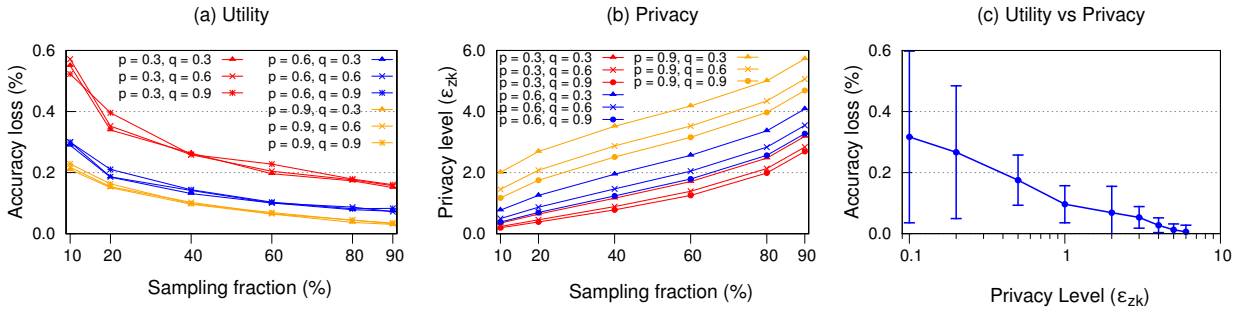

Figure 7: Results from the NYC Taxi case study with varying sampling and randomization parameters: **(a)** Utility, **(b)** Privacy level, **(c)** Comparison between utility and privacy.

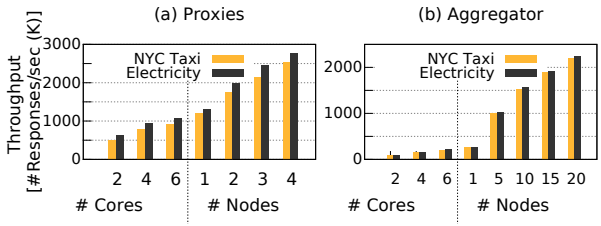

Figure 8: Throughput at proxies and the aggregator with different numbers of CPU cores and nodes.

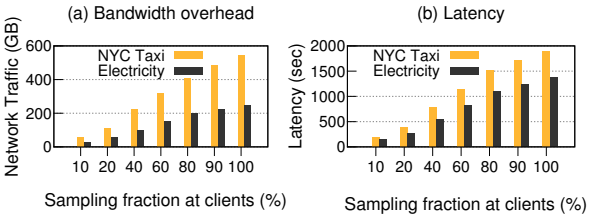

Figure 9: Total network traffic and latency at proxies with different sampling fractions at clients.

kWh, (0.5, 1] kWh, (1, 1.5] kWh, (1.5, 2] kWh, (2, 2.5] kWh, and (2.5, 3] kWh.

Evaluation metrics. We evaluated PRIVAPPROX using four key metrics: throughput, latency, utility, and privacy level. *Throughput* is defined as the number of data items processed per second, and *latency* is defined as the total amount of time required to process a certain dataset. *Utility* is the accuracy loss defined as $|\frac{estimate-exact}{exact}|$, where *estimate* and *exact* are the query results produced by applying PRIVAPPROX and the native computation, respectively. Finally, *privacy level* (ε_{zk}) is calculated using Equation 19 in the technical report [64]. For all measurements, we report the average over 10 runs.

7.2 Results from Case Studies

#I: Scalability. We measured the scalability of the two main system components: proxies and the aggregator. We first measured the throughput of proxies with different numbers of CPU cores (scale-up) and different numbers of nodes (scale-out). This experiment was conducted on a cluster of 4 nodes. Figure 8 (a) shows that, as expected, the throughput at proxies scales quite well with the number of CPU cores and nodes. In the NYC Taxi case study, with 2 cores, the throughput of each proxy is 512,348 answers/sec, and with 8 cores (1 node) the throughput is 1,192,903 answers/sec; whereas, with a cluster of 4 nodes each with 8 cores, the throughput of each proxy reaches 2,539,715 answers/sec. In the household electricity case study, the proxies achieve relatively higher throughput because the message size is smaller than in the NYC Taxi case study.

We next measured the throughput at the aggregator. Figure 8 (b) depicts that the aggregator also scales quite well when the number of nodes for aggregator increases. The throughput of the aggregator, however, is much lower than the throughput of proxies due to the relatively expensive `join` operation and the analytical computation at the aggregator. We notice that the throughput of the aggregator in the household electricity case study does not significantly improve in comparison to the first case study. This is because the difference in the size of messages between the two case studies does not affect much the performance of the `join` operation and the analytical computation.

#II: Network bandwidth and latency. Next, we conducted the experiment to measure the network bandwidth usage. By leveraging the sampling mechanism at clients, our system reduces network traffic significantly. Figure 9 (a) shows the total network traffic transferred from clients to proxies with different sampling fractions. In the first case study, with the sampling fraction of 60%, PRIVAPPROX can reduce the network traffic by 1.62×; whereas in the second case study, the reduction is 1.58×. Besides the benefit of saving network bandwidth, PRIVAPPROX also achieves lower latency in processing query answers by leveraging approximate computation. To evaluate this advantage, we measured the effect of sampling fractions on the latency of processing query answers. Figure 9 (b) depicts the latency with different sampling fractions at clients. For the first case study, with the sampling fraction of 60%, the latency is 1.68× lower than the execution without sampling; whereas, in the second case study, this value is 1.66× lower than the execution without sampling.

#III: Utility and privacy. Figure 7 (a)(b)(c) show the utility, the privacy level, and the trade-off between them,

respectively, with different sampling and randomization parameters. The randomization parameters p and q vary in the range of (0, 1), and the sampling parameter s is calculated using Equation 19 in the technical report [64]. Here, we show results only with the NYC Taxi dataset. As the sampling parameter s and the first randomization parameter p increase, the utility of query results improves (i.e., accuracy loss gets smaller) whereas the privacy guarantee gets weaker (i.e., privacy level gets higher). Since the NYC Taxi dataset is diverse, the accuracy loss and the privacy level change in a non-linear fashion with different sampling fractions and randomization parameters. Interestingly, the accuracy loss does not always decrease as the second randomization parameter q increases. The accuracy loss gets smaller when $q = 0.3$. This is due to the fact that the fraction of truthful "Yes" answers in the dataset is 33.57% (close to $q = 0.3$).

8 Related Work

Privacy-preserving analytics. Since the notion of differential privacy [32, 34], a plethora of systems have been proposed to provide differential privacy with centralized databases [46, 48, 52, 56–59, 63, 68]. In practice, however, such central trust can be abused, leaked, or subpoenaed [28, 49, 62, 69].

To overcome the limitations of the centralized database schemes, recently a flurry of systems have been proposed with a focus on preserving user privacy (mostly, differential privacy) in a distributed setting where the private data is kept locally [10, 26, 27, 33, 43, 44, 47, 55, 61, 71, 74]. However, these systems are designed to deal with the "one-shot" batch queries only, whereby the data is assumed to be static.

To overcome the limitations of the aforementioned systems, several differentially private stream analytics systems have been proposed [22, 23, 35, 38, 45, 66, 70]. Unfortunately, these systems still contain several technical shortcomings that limit their practicality. One of the first systems [35] updates the query result only if the user's private data changes significantly, and does not support stream analytics over an unlimited time period. Subsequent systems [23, 45] remove the limit on the time period, but introduce extra system overheads. Some systems [66, 70] leverage expensive secret sharing cryptographic operations to produce noisy aggregate query results. These systems, however, cannot work at large scale under churn; moreover, in these systems, even a single malicious user can substantially distort the aggregate results without detection. Recently, some other privacy-preserving distributed stream monitoring systems have been proposed [22, 38]. However, they all require some form of synchronization, and are tailored for heavy-hitter monitoring only. Streaming data publishing systems like [76] use a stream-privacy metric at the cost of relying on a trusted party to add noise. In contrast, PRIVAPPROX does not require a trusted proxy or aggregator to add noise. Furthermore, PRIVAPPROX provides stronger privacy properties (i.e., zero-knowledge privacy).

Sampling and randomized response. Sampling and randomized response, also known as input perturbation techniques, are being studied in the context of privacy-preserving analytics, albeit they are explored separately. For instance, the relationship between sampling and privacy is being investigated to provide k-anonymity [24], differential privacy [59], and crowd-blending privacy [40]. In contrast, we show that sampling combined with randomized response achieves the zero-knowledge privacy, a privacy bound strictly stronger than the state-of-the-art differential privacy.

Randomized response [37, 77] is a surveying technique in statistics, since 1960s, for collecting sensitive information via input perturbation. Recently, Google in a system called RAPPOR [73] made use of randomized response for privacy-preserving analytics. Like RAPPOR, PRIVAPPROX utilizes randomized response. However, RAPPOR is designed for heavy-hitter collection, and does not deal with the situation where clients' answers to the same query are changing over time. Therefore, RAPPOR does not fit well with the stream analytics. Furthermore, since we combine randomized response with sampling, PRIVAPPROX provides a privacy bound tighter than RAPPOR.

Approximate computation. Approximation techniques such as sampling [11, 25, 39], sketches [29], and online aggregation [50] have been well-studied over the decades in the databases community. Recently, sampling-based systems [8, 9, 42, 53, 65] have also been shown effective for "Big Data" analytics. We build on the advancements of sampling-based techniques. In particular, our work builds on IncApprox [53], a data analytics system that combines incremental computation [12, 15–18] and approximate computation. However, we differ in two crucial aspects. First, we perform sampling in a distributed way as opposed to sampling in a centralized dataset. Second, we extend sampling with randomized response for privacy-preserving analytics.

9 Conclusion

In this paper, we presented PRIVAPPROX, a privacy-preserving stream analytics system. Our approach builds on the observation that both computing paradigms — privacy-preserving data analytics and approximate computation — strive for approximation, and can be combined together to leverage the benefits of both. Our evaluation shows that PRIVAPPROX not only improves the performance to support real-time stream analytics, but also achieves provably stronger privacy guarantees than the state-of-the-art differential privacy. PRIVAPPROX's source code is publicly available: https://PrivApprox.github.io.

Acknowledgments. We thank anonymous reviewers and our shepherd Adam Bates for their helpful comments. This work is supported by the resilience path within CFAED at TU Dresden, the European Unions Horizon 2020 research and innovation programme under grant agreements 645011 (SERECA) and Amazon Web Services Education Grant.

References

[1] Apache S4. http://incubator.apache.org/s4. Accessed: May, 2017.

[2] Apache Spark Streaming. http://spark.apache.org/streaming. Accessed: May, 2017.

[3] Apache Storm. http://storm-project.net/. Accessed: May, 2017.

[4] Apache Zookeeper. https://zookeeper.apache.org/. Accessed: May, 2017.

[5] Kafka - A high-throughput distributed messaging system. http://kafka.apache.org. Accessed: May, 2017.

[6] Sample household electricity time of use data. https://goo.gl/0p2QGB. Accessed: May, 2017.

[7] SQLite. https://www.sqlite.org/. Accessed: May, 2017.

[8] Quickr: Lazily Approximating Complex Ad-Hoc Queries in Big Data Clusters. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2016.

[9] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *Proceedings of the ACM European Conference on Computer Systems (EuroSys)*, 2013.

[10] I. E. Akkus, R. Chen, M. Hardt, P. Francis, and J. Gehrke. Non-tracking web analytics. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2012.

[11] M. Al-Kateb and B. S. Lee. Stratified Reservoir Sampling over Heterogeneous Data Streams. In *Proceedings of the 22nd International Conference on Scientific and Statistical Database Management (SSDBM)*, 2010.

[12] P. Bhatotia. *Incremental Parallel and Distributed Systems*. PhD thesis, Max Planck Institute for Software Systems (MPI-SWS), 2015.

[13] P. Bhatotia, U. A. Acar, F. P. Junqueira, and R. Rodrigues. Slider: Incremental Sliding Window Analytics. In *Proceedings of the 15th International Middleware Conference (Middleware)*, 2014.

[14] P. Bhatotia, M. Dischinger, R. Rodrigues, and U. A. Acar. Slider: Incremental Sliding-Window Computations for Large-Scale Data Analysis. Technical Report MPI-SWS-2012-004, MPI-SWS, 2012. http://www.mpi-sws.org/tr/2012-004.pdf.

[15] P. Bhatotia, P. Fonseca, U. A. Acar, B. Brandenburg, and R. Rodrigues. iThreads: A Threading Library for Parallel Incremental Computation. In *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2015.

[16] P. Bhatotia, R. Rodrigues, and A. Verma. Shredder: GPU-Accelerated Incremental Storage and Computation. In *Proceedings of USENIX Conference on File and Storage Technologies (FAST)*, 2012.

[17] P. Bhatotia, A. Wieder, I. E. Akkus, R. Rodrigues, and U. A. Acar. Large-scale incremental data processing with change propagation. In *Proceedings of the Conference on Hot Topics in Cloud Computing (HotCloud)*, 2011.

[18] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar, and R. Pasquini. Incoop: MapReduce for Incremental Computations. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2011.

[19] A. Blum, C. Dwork, F. McSherry, and K. Nissim. Practical privacy: the SuLQ framework. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, 2005.

[20] A. Blum, K. Ligett, and A. Roth. A Learning Theory Approach to Non-interactive Database Privacy. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2008.

[21] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 2015.

[22] T.-H. H. Chan, M. Li, E. Shi, and W. Xu. Differentially Private Continual Monitoring of Heavy Hitters from Distributed Streams. In *Proceedings of the 12th International Conference on Privacy Enhancing Technologies (PETS)*, 2012.

[23] T.-H. H. Chan, E. Shi, and D. Song. Private and Continual Release of Statistics. *ACM Trans. Inf. Syst. Secur.*, 2011.

[24] K. Chaudhuri and N. Mishra. When Random Sampling Preserves Privacy. In *Proceedings of the 26th Annual International Conference on Advances in Cryptology (CRYPTO)*, 2006.

[25] S. Chaudhuri, G. Das, and V. Narasayya. Optimized Stratified Sampling for Approximate Query Processing. *Proceedings of ACM Transaction of Database Systems (TODS)*, 2007.

[26] R. Chen, I. E. Akkus, and P. Francis. SplitX: High-performance Private Analytics. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2013.

[27] R. Chen, A. Reznichenko, P. Francis, and J. Gehrke. Towards Statistical Queries over Distributed Private User Data. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012.

[28] ComScore Reaches $14 Million Settlement in Electronic Privacy Class Action. http://www.alstonprivacy.com/comscore-reaches-14-million-settlement-in-electronic-privacy-class-action/. Accessed: May, 2017.

[29] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Found. Trends databases*, 2012.

[30] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.

[31] J. R. Douceur. The Sybil Attack. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[32] C. Dwork. Differential privacy. In *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming, part II (ICALP)*, 2006.

[33] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our Data, Ourselves: Privacy Via Distributed Noise Generation. In *Proceedings of the 24th Annual International Conference on The Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2006.

[34] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In *Proceedings of the Third conference on Theory of Cryptography (TCC)*, 2006.

[35] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum. Differential privacy under continual observation. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2010.

[36] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[37] J. A. Fox and P. E. Tracy. *Randomized response: a method for sensitive surveys*. Beverly Hills California Sage Publications, 1986.

[38] A. Friedman, I. Sharfman, D. Keren, and A. Schuster. Privacy-Preserving Distributed Stream Monitoring. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*, 2014.

[39] M. N. Garofalakis and P. B. Gibbon. Approximate Query Processing: Taming the TeraBytes. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2001.

[40] J. Gehrke, M. Hay, E. Lui, and R. Pass. Crowd-blending privacy. In *Proceedings of the 32nd Annual International Conference on Advances in Cryptology (CRYPTO)*, 2012.

[41] J. Gehrke, E. Lui, and R. Pass. Towards Privacy for Social Networks: A Zero-Knowledge Based Definition of Privacy. In *Theory of Cryptography*, 2011.

[42] I. Goiri, R. Bianchini, S. Nagarakatte, and T. D. Nguyen. ApproxHadoop: Bringing Approximations to MapReduce Frameworks. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2015.

[43] S. Guha, B. Cheng, and P. Francis. Privad: Practical Privacy in Online Advertising. In *Proceedings of the 8th Symposium on Networked Systems Design and Implementation (NSDI)*, 2011.

[44] S. Guha, M. Jain, and V. N. Padmanabhan. Koi: A location-privacy platform for smartphone apps. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012.

[45] T. h. Hubert Chan, E. Shi, and D. Song. Privacy-preserving stream aggregation with fault tolerance. In *Proceedings of 16th International Conference on Financial Cryptography and Data Security (FC)*, 2012.

[46] A. Haeberlen, B. C. Pierce, and A. Narayan. Differential Privacy Under Fire. In *Proceedings of the 20th USENIX Security Symposium (USENIX Security)*, 2011.

[47] M. Hardt and S. Nath. Privacy-aware personalization for mobile advertising. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS)*, 2012.

[48] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the Accuracy of Differentially Private Histograms Through Consistency. *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2010.

[49] HealthCare.gov Sends Personal Data to Dozens of Tracking Websites. `https://www.eff.org/deeplinks/2015/01/healthcare.gov-sends-personal-data`. Accessed: May, 2017.

[50] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online Aggregation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 1997.

[51] Z. Jerzak and H. Ziekow. The debs 2015 grand challenge. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems (DEBS)*, 2015.

[52] V. Karwa, S. Raskhodnikova, A. Smith, and G. Yaroslavtsev. Private analysis of graph structure. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2011.

[53] D. R. Krishnan, D. L. Quoc, P. Bhatotia, C. Fetzer, and R. Rodrigues. IncApprox: A Data Analytics System for Incremental Approximate Computing. In *Proceedings of International Conference on World Wide Web (WWW)*, 2016.

[54] J. Lee and C. Clifton. How Much is Enough? Choosing ε for Differential Privacy. In *Proceedings of the 14th International Conference on Information Security (ISC)*, 2011.

[55] S. Lee, E. L. Wong, D. Goel, M. Dahlin, and V. Shmatikov. πBox: A Platform for Privacy-Preserving Apps. In *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.

[56] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing Linear Counting Queries Under Differential Privacy. In *Proceedings of the ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, 2010.

[57] F. McSherry. Privacy Integrated Queries. In *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2009.

[58] F. McSherry and R. Mahajan. Differentially-private Network Trace Analysis. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2010.

[59] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. Culler. GUPT: Privacy Preserving Data Analysis Made Easy. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2012.

[60] D. S. Moore. *The Basic Practice of Statistics*. W. H. Freeman & Co., 2nd edition, 1999.

[61] A. Narayan and A. Haeberlen. DJoin: Differentially Private Join Queries over Distributed Databases. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2012.

[62] Privacy Lawsuit Targets Net Giants Over 'Zombie' Cookies. `http://www.wired.com/2010/07/zombie-cookies-lawsuit`. Accessed: May, 2017.

[63] D. Proserpio, S. Goldberg, and F. McSherry. Calibrating Data to Sensitivity in Private Data Analysis: A Platform for Differentially-private Analysis of Weighted Datasets. *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2014.

[64] D. L. Quoc, M. Beck, P. Bhatotia, R. Chen, C. Fetzer, and T. Strufe. Privacy preserving stream analytics: The marriage of randomized response and approximate computing. `https://arxiv.org/abs/1701.05403`, 2017.

[65] D. L. Quoc, R. Chen, P. Bhatotia, C. Fetzer, V. Hilt, and T. Strufe. StreamApprox: Approximate Computing for Stream Analytics. 2017.

[66] V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In

Proceedings of the International Conference on Management of Data (SIGMOD), 2010.

[67] M. G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 1998.

[68] I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and Privacy for MapReduce. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2010.

[69] SEC Charges Two Employees of a Credit Card Company with Insider Trading. `http://www.sec.gov/litigation/litreleases/2015/lr23179.htm`. Accessed: May, 2017.

[70] E. Shi, T. H. Chan, E. G. Rieffel, R. Chow, and D. Song. Privacy-Preserving Aggregation of Time-Series Data. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*, 2011.

[71] K. Singh, S. Bhola, and W. Lee. xbook: Redesigning privacy control in social networking platforms. In *Proceedings of the 18th Conference on USENIX Security Symposium (USENIX Security)*, 2009.

[72] S. K. Thompson. *Sampling*. Wiley Series in Probability and Statistics, 2012.

[73] E. Úlfar, P. Vasyl, and K. Aleksandra. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014.

[74] B. Viswanath, E. Kiciman, and S. Saroiu. Keeping Information Safe from Social Networking Apps. In *Proceedings of the ACM SIGCOMM Workshop on Social Networks (WOSN'12)*, 2012.

[75] G. Wang, B. Wang, T. Wang, A. Nika, H. Zheng, and B. Y. Zhao. Defending against sybil devices in crowdsourced mapping services. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2016.

[76] Q. Wang, Y. Zhang, X. Lu, Z. Wang, Z. Qin, and K. Ren. RescueDP: Real-time Spatio-temporal Crowd-sourced Data Publishing with Differential Privacy. In *Proceedings of the 35th Annual IEEE International Conference on Computer Communications (INFOCOM)*, 2016.

[77] S. L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. In *Journal of the American Statistical Association*, 1965.

[78] A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues. Brief Announcement: Modelling MapReduce for Optimal Execution in the Cloud. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of Distributed Computing (PODC)*, 2010.

[79] A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues. Conductor: Orchestrating the Clouds. In *Proceedings of the 4th international workshop on Large Scale Distributed Systems and Middleware (LADIS)*, 2010.

[80] A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues. Orchestrating the Deployment of Computations in the Cloud with Conductor. In *Proceedings of the 9th USENIX symposium on Networked Systems Design and Implementation (NSDI)*, 2012.