# Poster: Generating Reproducible Out-of-Order Data Streams

Philipp M. Grulich[1] Jonas Traub[1,2] Asterios Katsifodimos[3] Tilmann Rabl[4*]
Sebastian Breß[1,2] Volker Markl[1,2]
[1]Technische Universitat Berlin [2]DFKI GmbH [3]Delft University of Technology [4]Hasso Plattner Institute

## ABSTRACT

Evaluating modern stream processing systems in a reproducible manner requires data streams with different data distributions, data rates, and real-world characteristics such as delayed and out-of-order tuples. In this paper, we present an open source stream generator which generates reproducible and deterministic out-of-order streams based on real data files, simulating arbitrary fractions of out-of-order tuples and their respective delays.

## CCS CONCEPTS

• **Information systems → Stream management**.

## KEYWORDS

Stream Processing, Benchmarking, Data Generation, Out-of-Order

**ACM Reference Format:**
Philipp M. Grulich, Jonas Traub, Asterios Katsifodimos, Tilmann Rabl, Sebastian Breß, and Volker Markl. 2019. Poster: Generating Reproducible Out-of-Order Data Streams. In *DEBS '19: The 13th ACM International Conference on Distributed and Event-based Systems (DEBS '19), June 24–28, 2019, Darmstadt, Germany.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3328905.3332511

## 1 INTRODUCTION

With the growing adoption of stream processing systems (SPSs), it is crucial to evaluate these systems on reproducible and realistic workloads [5]. To this end, a variety of real-world streaming data sets from different domains are available, such as the NYC TLC Trip Record Data [9], the DEBS Grand Challenge data sets [4, 7], and the UCI Human Activity Recognition data set [2]. An important characteristic of data streams is the order of events: the order of events arriving at a stream processing system does not always match the event-time (the point in time at which an individual event was generated). Many factors can affect the order of events, such as slow data sources, clock drifts, or network disconnects and delays. If the stream processor does not take this out-of-orderness into account, it can generate incorrect and non-deterministic query results. $I^2$, for example, relies on the correct event order to visualize time series [12]. To handle out-of-order events directly in the SPSs many systems follow the Dataflow Model [1] and implement the bucket-per-window technique [6]. As processing performance is crucial for SPSs, recent works focused on efficient aggregation techniques based on aggregate sharing [3, 8, 10, 11]. To evaluate these techniques and the impact of out-of-order handling on the system performance, it is necessary to modify real-world data sets in order to meet specific data characteristics such as out-of-orderness and the addition of delay in event arrivals.

[*]Work condcuted while the author was employed at TU Berlin

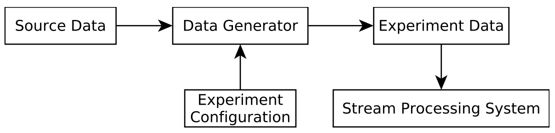

**Figure 1: Overview of data generation.**

In this paper, we present an out-of-order stream data generator, which enables the reproducible modification of out-of-order characteristics of arbitrary input data sets. This tool was already used in recent work on efficient window aggregation with General Stream Slicing [10, 11] and is available in a public repository.[1] The data generator supports benchmark developers in two aspects. First, it analyzes the out-of-order characteristics of a given source data set, such as the fraction of out-of-order events and the minimal/maximal event delay. Second, the data generator modifies the out-of-order factor of the source data set by introducing delays to specific events. For this, the generator ensures that the temporal data distribution remains constant. This guarantees that the results of an event time stream processing query are independent of the introduced delay. To enable reproducibility, given a configuration, we generate experiment data on the fly deterministically.

This paper describes the architecture of our out-of-order data generator, the design of its configuration files, and its out-of-order manipulation algorithm. Furthermore, the goal of this poster is threefold. First, we want to raise attention for our data generator in the stream processing community at DEBS. Second, we wish to gather contributions to our open source data generator. Finally, we aim to establish a novel benchmark for out-of-order stream processing; our generator provides a first step towards that direction.

## 2 DATA GENERATOR

Figure 1 shows the overall data generation process, from the source data to the final experiment data. Given identical source and configuration files, our generator guarantees to produce the same experiment data to enable reproducibility. Finally, the generated data set can then be ingested into any SPS. In the following, we describe the source data, the configuration file, and the out-of-order data generation algorithm in detail.

### 2.1 Reusing Existing Streaming Datasets

For the evaluation of SPSs, we usually replay a set of prerecorded events from a source data file. The events can originally be generated by any kind of data sources such as sensors or real-time

[1]https://github.com/TU-Berlin-DIMA/out-of-order-datagenerator

```
"dataSource": {
  "file": $path$,
  "seperator": ","|";"|"\t",
  "time": {
    "timeIndex": $field$,
    "sourceTimeUnit": "ps"|"ns"|"ms"|"s"
  }
},
"experimentDataConfigurations": [
  {
    "targetOutOfOrderFactor": [0-100],
    "minDelay": 0,
    "maxDelay": 2000,
    "delaySeed": $seed$
  }
]
```

**Listing 1: Configuration file for data generator.**

applications. Our data generator can process any source file as long as it is encoded in a flat data format, and its events contain a field with the event time. Event-time is necessary to reason about the order and frequency of events.

## 2.2 Configuring the Data Generator

To generate experiment data we modify the source data set according to a configuration file, similar to the one in Listing 1. The configuration file is structured in two segments. First, it describes the data source set with its path, its field separator, and how the event time can be extracted. For this, we support time-units from picoseconds to seconds. The second component defines a list of experiment data configurations. Every configuration generates a separate experiment data set based on the same source data set. An experiment data configuration defines the targeted out-of-order factor and the minimal, maximal delay of an event, and an initial seed to initialize the delay generator.

## 2.3 Adding deterministic out-of-orderness

It is crucial to follow a deterministic algorithm that generates experiment data, changing delays and the fraction of out-of-order events. A naïve out-of-order data generation algorithm simply adds a random delay in the event time of certain events. This approach introduces out-of-orderness, but it also changes the temporal data distribution on the event time. For instance, a given event can be assigned to a different event time window, compared to a previously generated dataset. To mitigate this problem we propose the following three-phase data generation algorithm.

*2.3.1 Preprocessing.* In this step, we analyze the out-of-order factor of the source file, looking for already existing out-of-order events. This step is crucial: if a user wishes to alter the out-of-orderness of a given stream, our generator needs to know in advance what the characteristics of the original dataset are. To this end, our generator builds histograms and statistics on the temporal distribution of out-of-order events, in order to take those into account during the data generation step.

*2.3.2 Generation of out-of-order ingestion time.* Instead of modifying the event-time of an event, we pre-define a target ingestion time as a separate field of the input event. Initially, the ingestion time equals the event's event-time and prescribes *when an event should be consumed* by the SPSs. If a given event is already out-of-order in the source data set, we will just copy it to the generated data set. For every source in-order event, we decide if it will be transformed into an out-of-order event based on the out-of-order factor, specified by the user in the configuration file. To transform an in-order event to an out-of-order event, we alter its ingestion time: the existing event-time plus an out-of-order delay. In its current version, delays are uniformly distributed between the minimal and maximal delay as defined in the configuration file. However, it would be simple to plugin different distributions in our existing codebase.

*2.3.3 Sorting the final data stream by Ingestion Time.* After generating the target ingestion time for every event, the resulting data stream is still in the same order as the original stream. For existing SPSs to take advantage of the out-of-orderness without any changes in how they ingest streams, the generator sorts the resulting data stream by the event's ingestion time (the one that the generator altered to match the configuration's specification). The resulting dataset can simply be ingested by a given SPS, in the order that the events appear in the file. Care has to be taken though, that during the data ingestion the delays of each event are respected.

## 2.4 Analyzing Existing Streams

Apart from generating experiment data, it is often necessary to analyze the out-of-order properties of a given stream. To this end, we provide a second tool, which just performs the preprocessing phase of the data generation algorithm. This analysis step extracts the frequency, out-of-order factor, and a delay histogram.

## ACKNOWLEDGMENTS

This work was funded by the EU projects E2Data (780245), DFG Priority Program "Scalable Data Management for Future Hardware" (MA4662-5), and the German Ministry for Education and Research as BBDC II (01IS18025A).

## REFERENCES

[1] Tyler Akidau, Robert Bradshaw, et al. 2015. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. In *VLDB*.
[2] Davide Anguita, Alessandro Ghio, Luca Oneto, et al. 2013. A public domain dataset for human activity recognition using smartphones.. In *Esann*.
[3] Savong BOU, Hiroyuki KITAGAWA, and Toshiyuki AMAGASA. 2018. CBiX: Incremental Sliding-Window Aggregation For Real-Time Analytics Over Out-of-Order Data Streams. In *DEIM*.
[4] Zbigniew Jerzak, Thomas Heinze, Matthias Fehr, et al. 2012. The DEBS 2012 Grand Challenge. In *DEBS*.
[5] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen, and Volker Markl. 2018. Benchmarking distributed stream data processing systems. In *ICDE*.
[6] Jin Li, David Maier, Kristin Tufte, et al. 2005. Semantics and evaluation techniques for window aggregates in data streams. In *SIGMOD*.
[7] Christopher Mutschler, Holger Ziekow, and Zbigniew Jerzak. 2013. The DEBS 2013 Grand Challenge. In *DEBS*.
[8] Kanat Tangwongsan, Martin Hirzel, and Scott Schneider. 2018. Sub-O (log n) Out-of-Order Sliding-Window Aggregation. *arXiv preprint* (2018).
[9] New York City Taxi and Limousine Commission. [n.d.]. Tlc trip record data. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page.
[10] Jonas Traub, Philipp M. Grulich, Alejandro R. Cuéllar, Sebastian Breß, Asterios Katsifodimos, Tilmann Rabl, and Volker Markl. 2018. Scotty: Efficient Window Aggregation for Out-of-Order Stream Processing. In *ICDE*.
[11] Jonas Traub, Philipp M. Grulich, Alejandro R. Cuéllar, Sebastian Breß, Asterios Katsifodimos, Tilmann Rabl, and Volker Markl. 2019. Efficient Window Aggregation with General Stream Slicing. In *EDBT*.
[12] Jonas Traub, Nikolaas Steenbergen, Philipp Grulich, Tilmann Rabl, and Volker Markl. 2017. I2: Interactive Real-Time Visualization for Streaming Data.. In *EDBT*.