# A Synchrophasor Stream Processing Pipeline Architecture for Near-Real-Time Applications

MSc. Daniel Villegas

Dr. Athula Rajapakse

Electrical and Computer Engineering
University of Manitoba
2025

# Overview

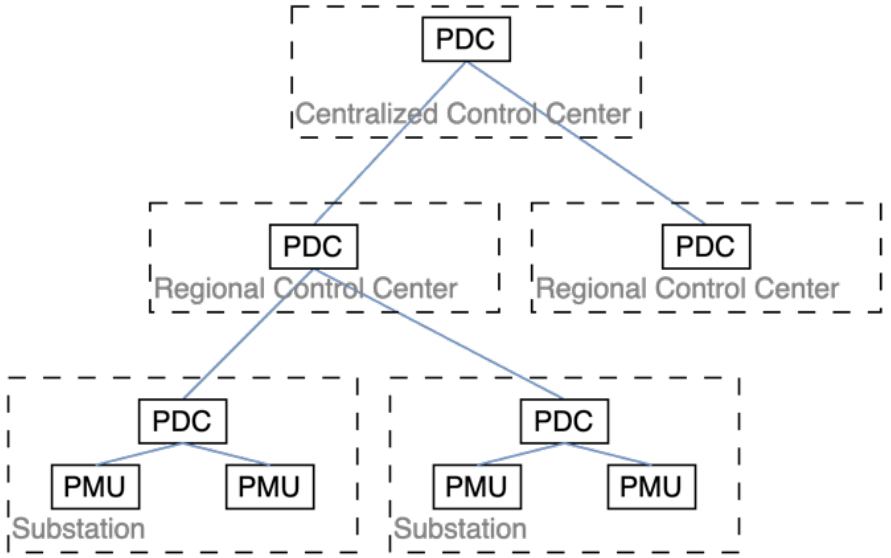
# Introduction

# Introduction

- Synchrophasor systems are wide area sensor networks dedicated to take phasor power system measurements.
- Synchrophasor networks were introduced in the 1990s as a modern alternative to SCADA systems.
- Their introduction was motivated by the increase in measurement time resolution and the GPS time synchronization.

# Traditional Synchrophasor Network

- A hierarchical synchrophasor network consists of multiple layers of PDC that forward data from Phasor Measurement Units (PMU) data to the corresponding PDC.
- Defined in NASPInet 1.0 Architecture Guidance and IEEE Std C37.118.2-2011 [2]
- Requires expensive and dedicated network infrastructure
-



# Problem Definition

- Except for redundancy, PDC are currently single node systems
- The system has limited horizontal scalability due to the hierarchical nature
- Existing applications often rely on proprietary hardware that may pose a challenge when integrating with popular big-data frameworks.
- This research investigates the technical aspects of implementing a cloud-based synchrophasor processing pipeline to enable the creation of near-real-time applications using open-source technologies.

# Objectives

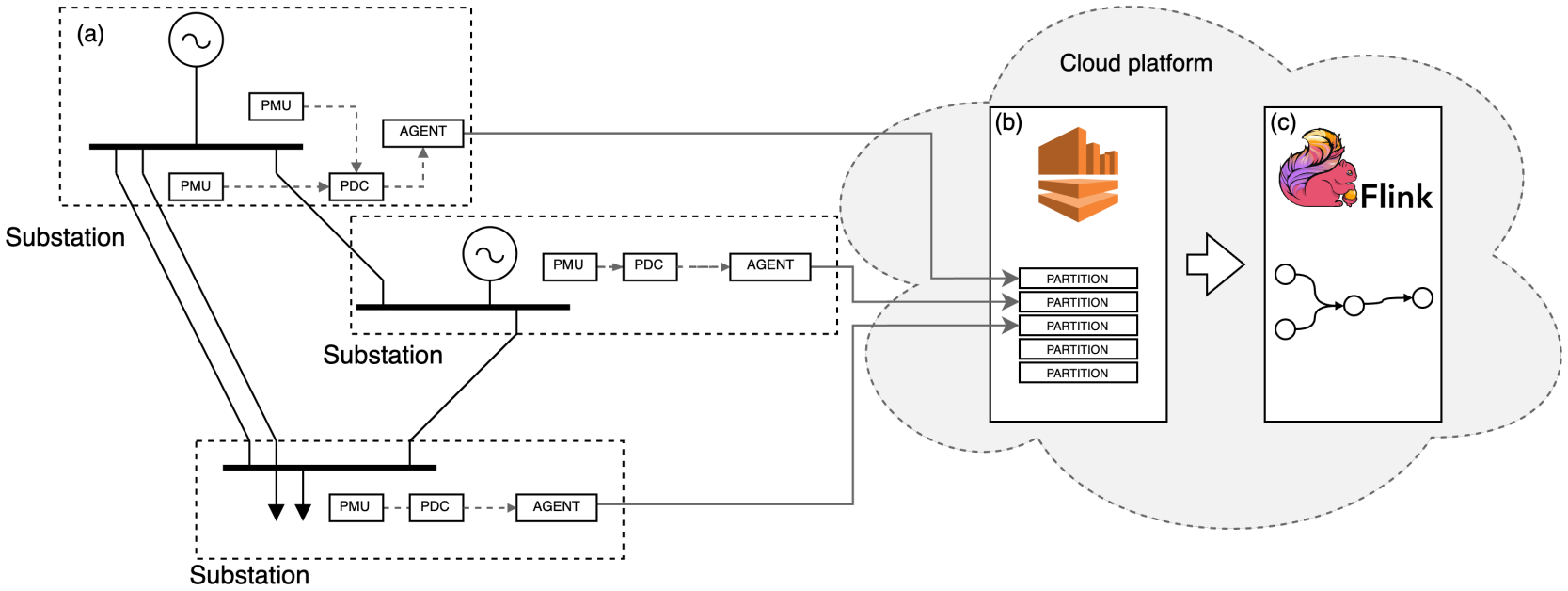
## Primary Objectives

- To develop a cloud-based synchrophasor data platform capable of ingesting data from multiple substations.

- To implement a generic reconfigurable stream processing pipeline where near-real-time synchrophasor applications can be implemented.

# Synchrophasor Processing Cloud Platform

# Cloud Platform Overview

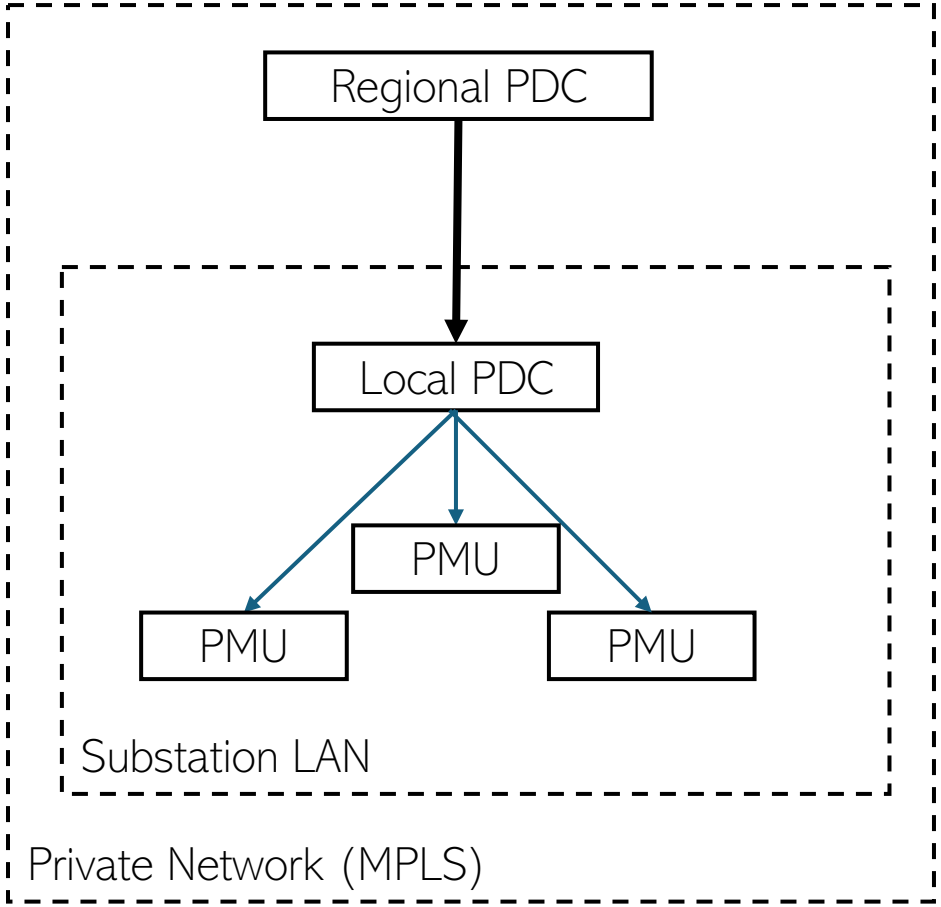



(a) Substation with PMU/PDC data and the agent that collects the data and initiates the connection to the cloud

(b) Distributed Message Broker to ingest the data on the cloud (Amazon Kinesis)

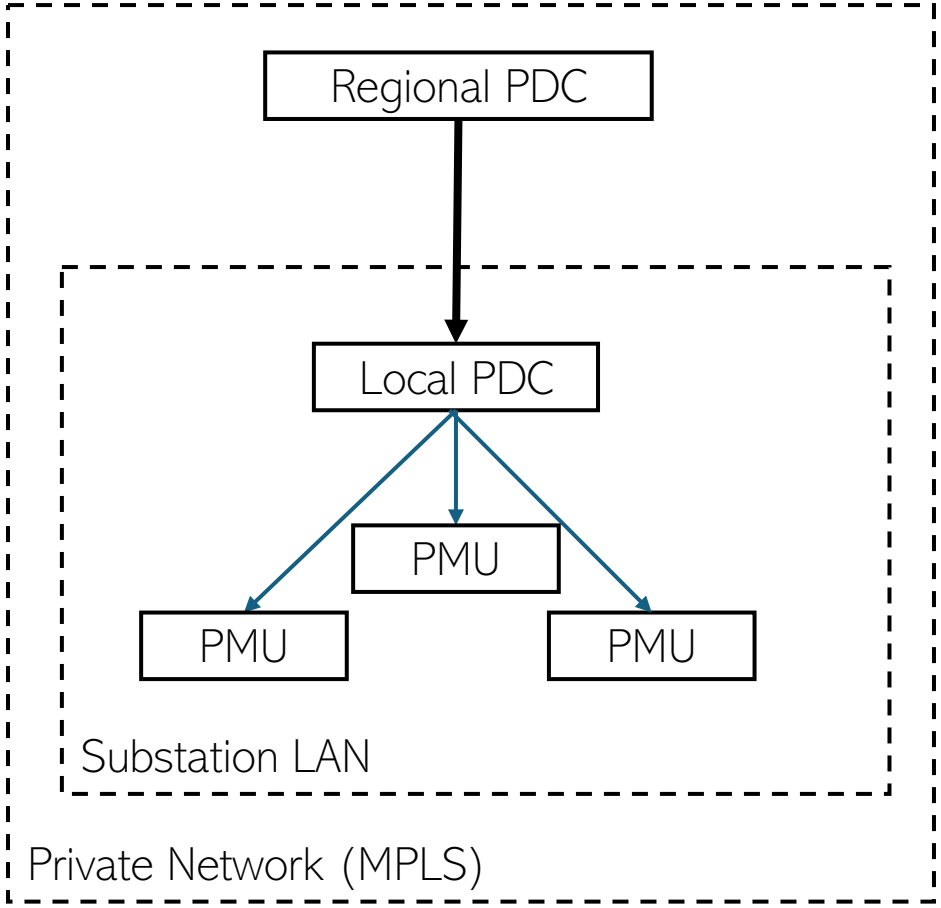
(c) Synchrophasor Data Processing Pipeline built using Apache Flink

# Substation Agent

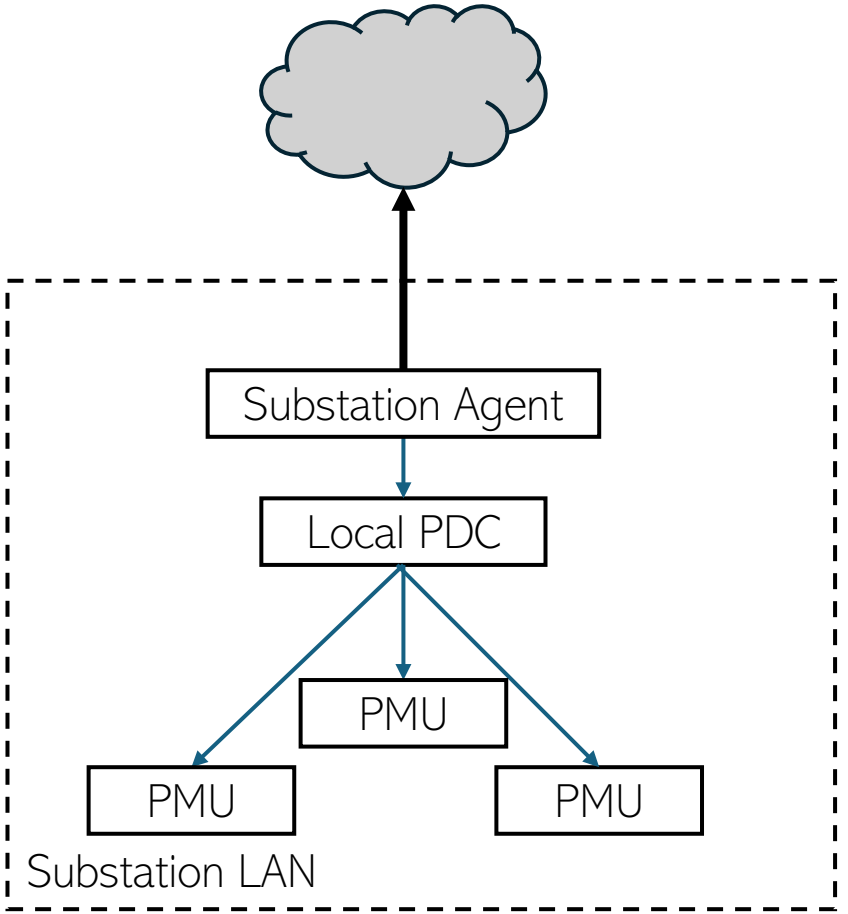



Typical Approach

# Substation Agent

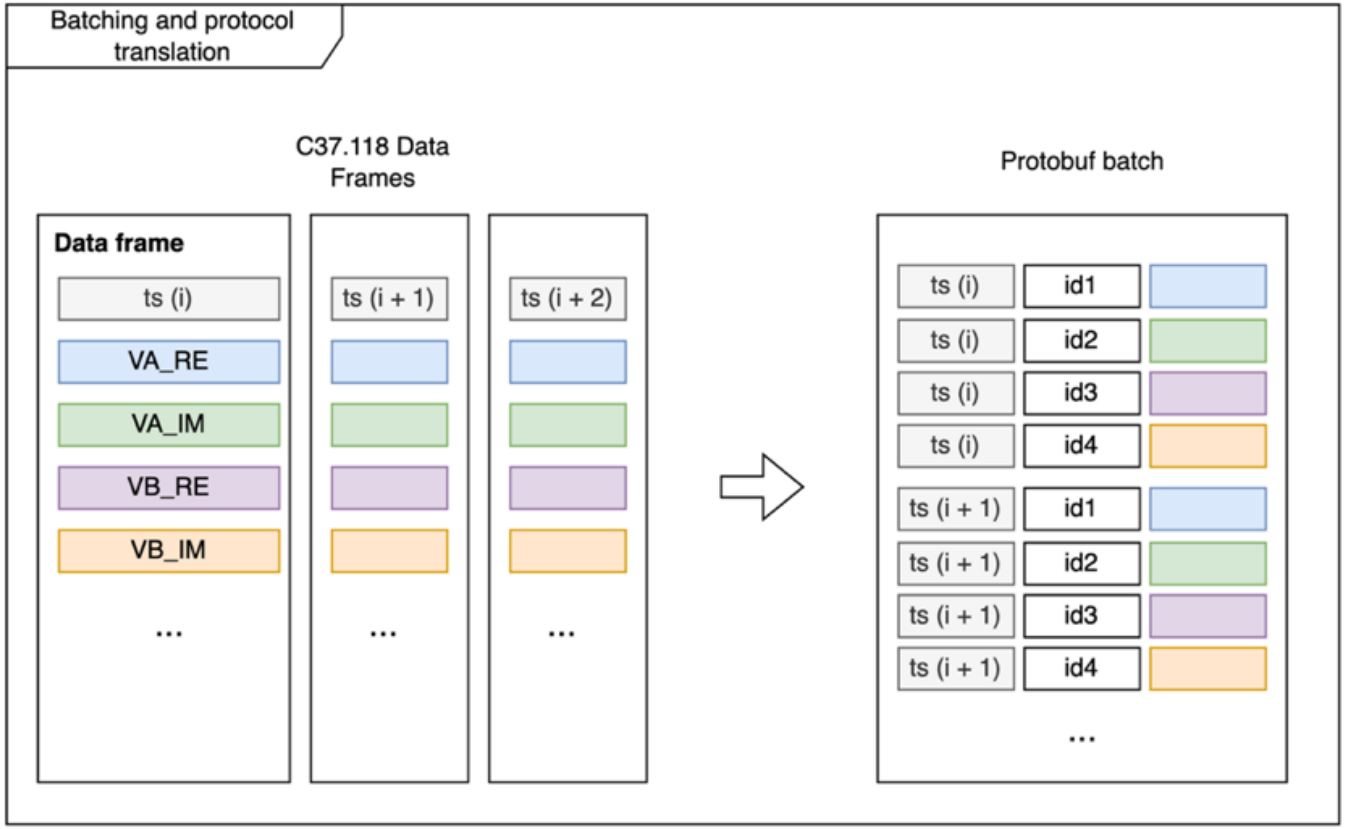



Typical Approach

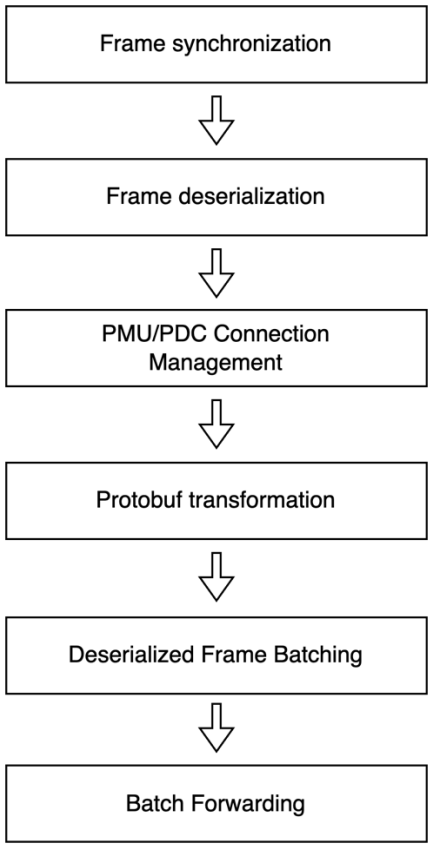



Proposed Approach

# Substation Agent



# Substation Agent




Frame synchronization detects the beginning of the frame and discards incomplete or corrupted data that may be on the buffer.

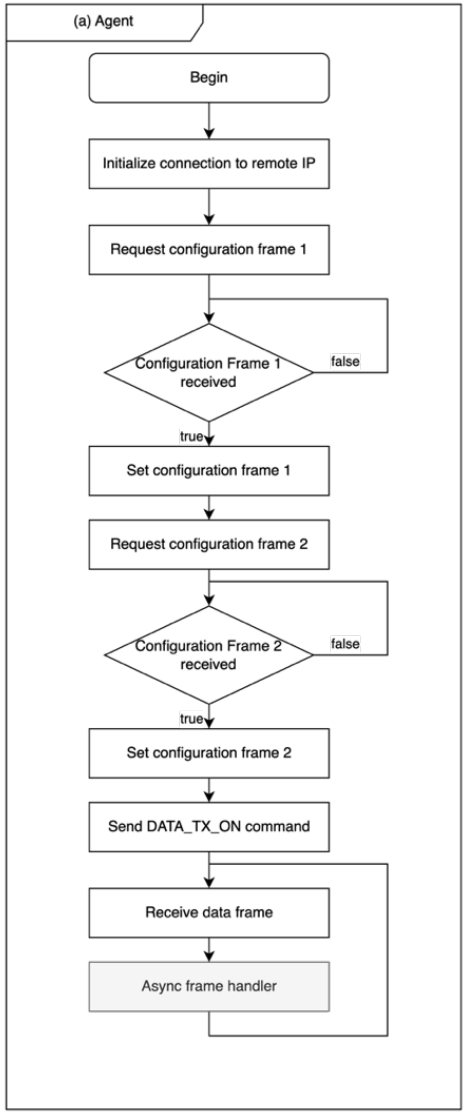
Once a frame is ultimately received, it is deserialized into an object.

Connection Management establishes the connection, manages its state and retries the connection on failure.

Received measurements are transformed to Protobuf for being transmitted to the cloud.

Protouf frames are batched.

When a batch is created, it is forwarded to the cloud.




# Substation Agent

## Connection Management

**1.** The agent connects to a preconfigured IP address and requests the configuration frame 1

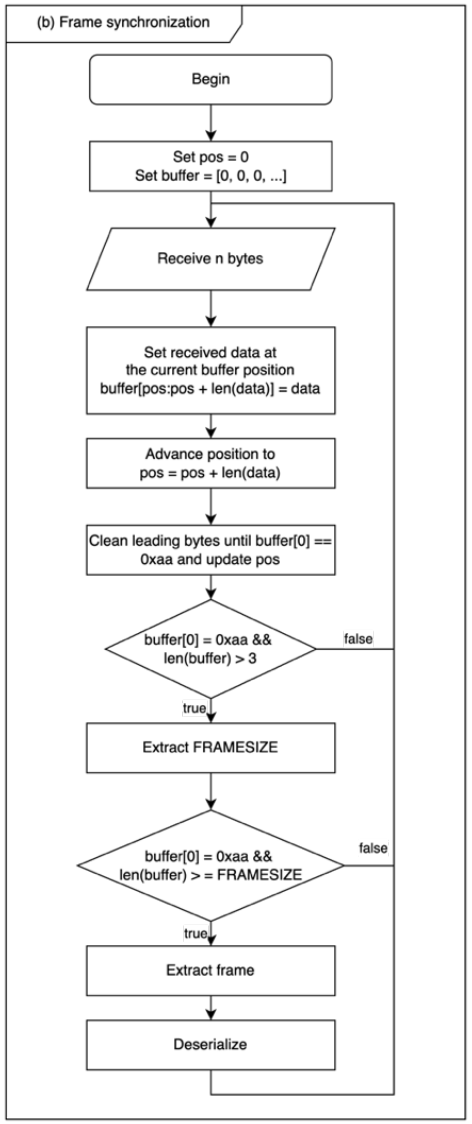
**2.** Once this is received, it is set in an internal variable

**3.** The agent requests configuration frame 2

**4.** The configuration frame 2 is set

**5.** The agent sends a command to initiate data transmission

**6.** Data frames are received continuously and sent to an asynchronous frame handler

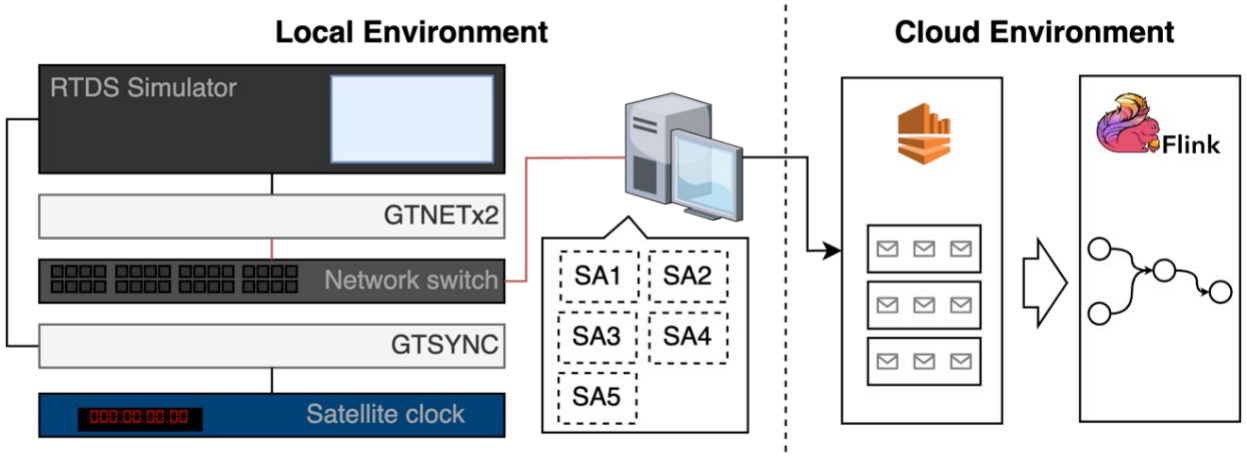



# Substation Agent

## Frame Synchronization

1. The position of the reception buffer is set to zero
2. N bytes are received, and the position of the buffer is updated
3. Seek for the first synchronization byte (0xAA) and discard leading bytes (
4. Repeat until more than 3 bytes are in the buffer (2)
5. Extract the frame size and repeat until the whole frame is received (2)
6. Extract the frame
7. Deserialize
8. Repeat the process for the next frame (2)

# Experimental Setup

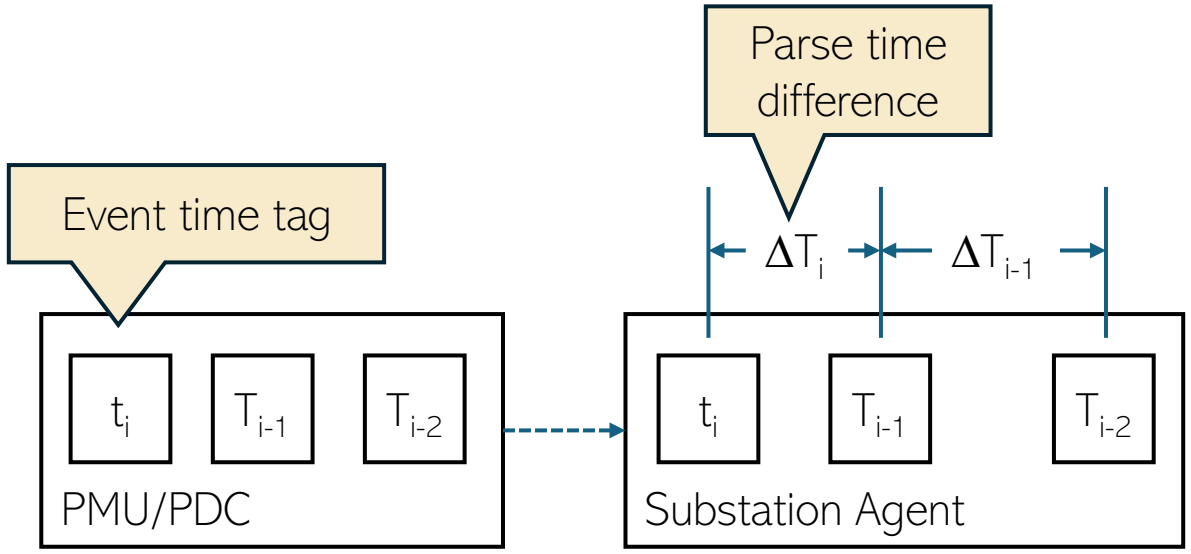



- The power system is simulated using a real-time simulator with PMU emulation.
- The substation agents are hosted in a computer in the lab.
- The data is forwarded as it is received to the cloud platform.
- Lastly, the data is consumed by the synchrophasor data pipeline.

# Substation Agent

## Validation

Four experiments were conducted at 10, 30 and 60 frames per second. The 30 frame per second experiment was conducted twice to verify the same results were obtained




To validate the substation agent's correct functioning, the time difference between subsequent frames was measured using the computer's monotonic clock. This time is expected to remain constant

$\Delta T$ is the time difference between consecutive frames

$t_{parse}$ Are the timestamps measured on when the frames are parsed and ready to be batched and forwarded.

$$\Delta T = t_{parse,i} - t_{parse,i-1}$$

# Substation Agent

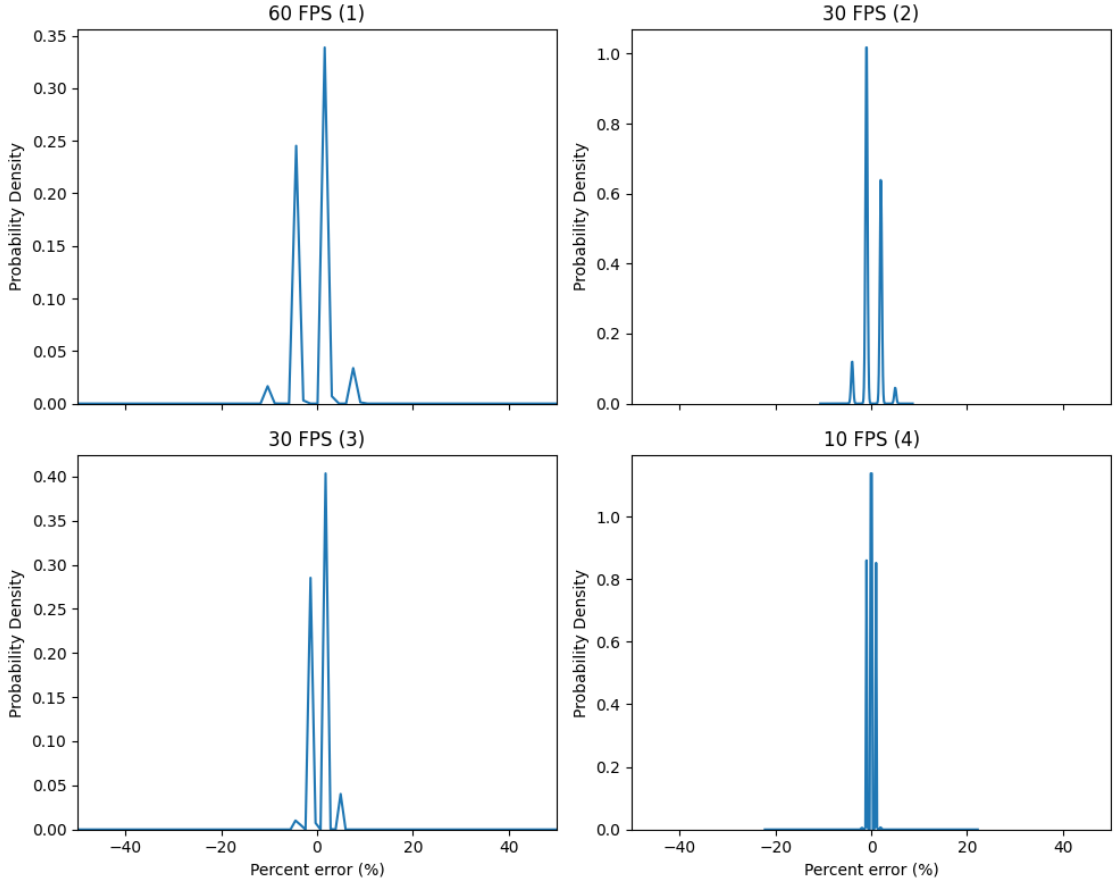
## Expected period validation

- $\epsilon$ Is the difference between the expected period and the measured period, which is the inverse of the data rate
- $\delta$ is the normalized difference between the expected period and the measured period.

$$\epsilon = \Delta T - \frac{1}{r}$$

$$\delta = \frac{\left(\Delta T - \frac{1}{r}\right)}{\frac{1}{r}}$$

- The histograms show the probability density of the normalized error measured



# Substation Agent

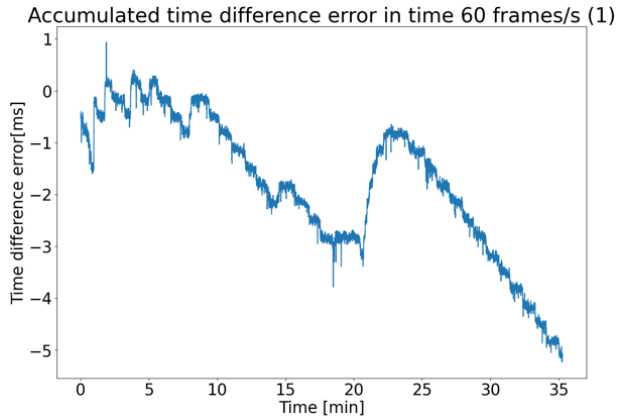
## Accumulated drift validation

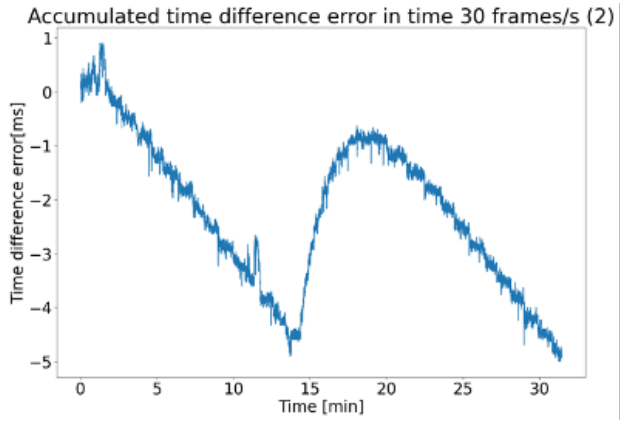
- Since the time differences are relatively small (<10ms), the accumulated error of the error in the expected arrival period is calculated during the experiment to assess the long-term performance of the program

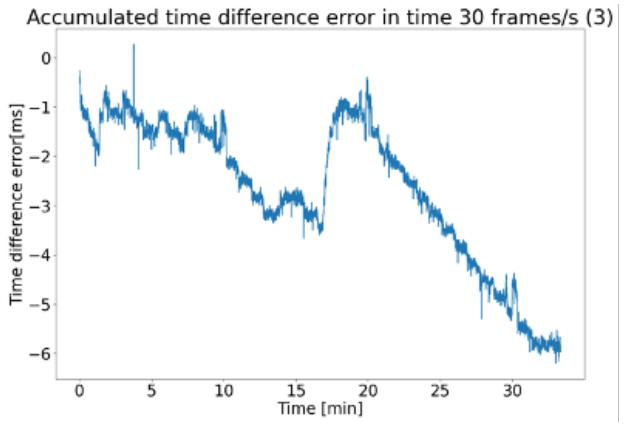
$$\Delta E = \sum_{i=0}^{n-1} \Delta \epsilon_i$$

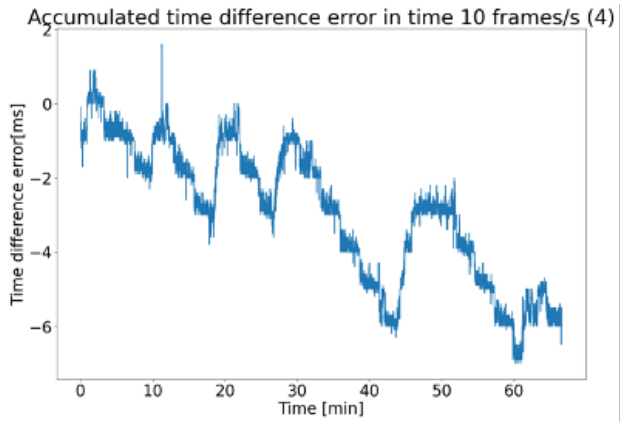
- If working correctly, the accumulated error is expected to be around zero or small relative to the period.

- The results show a very small negative drift (6ms during 1 hour in the worst case). This seems to be attributable to the natural drift of the clock which is 27 ms every 30 min.



# Cloud Infrastructure

The data pipeline, distributed broker and other supporting infrastructure is shown.

(1) Load balancer for the user to interact with the metadata and operations registry.

(2) A metadata and operations registry database exposed through a web and rest interfaces.

(3) An Amazon Kinesis Streams to receive incoming data from the substation agent

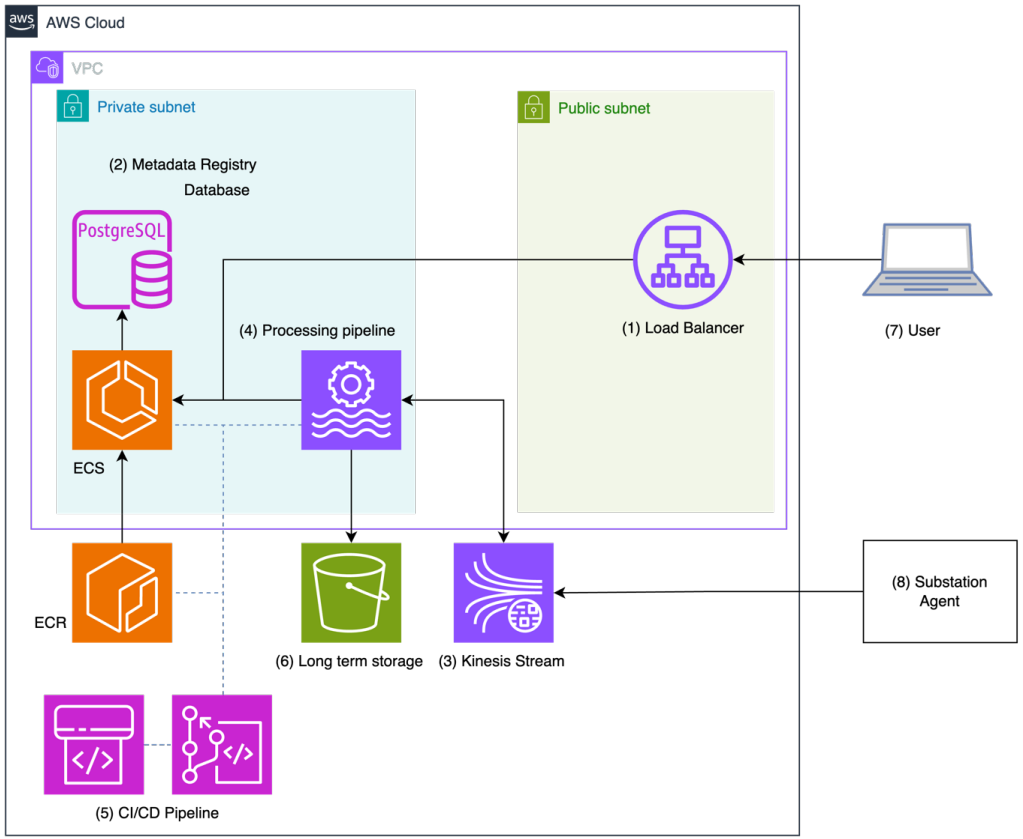
(4) Managed Apache Flink cluster to host the processing pipeline

(5) CI/CD tools to automate deployments

(6) Long term storage bucket

(7) User that interacts with the database

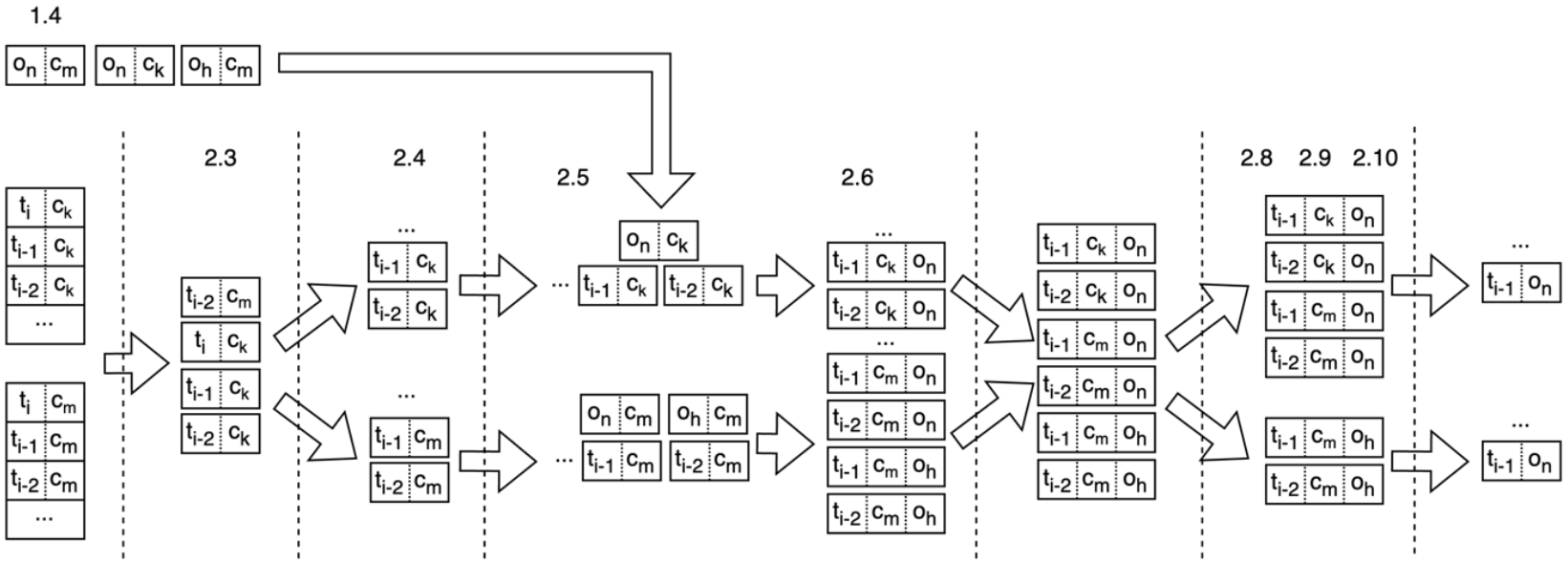
(8) Substation agent



# Processing Pipeline

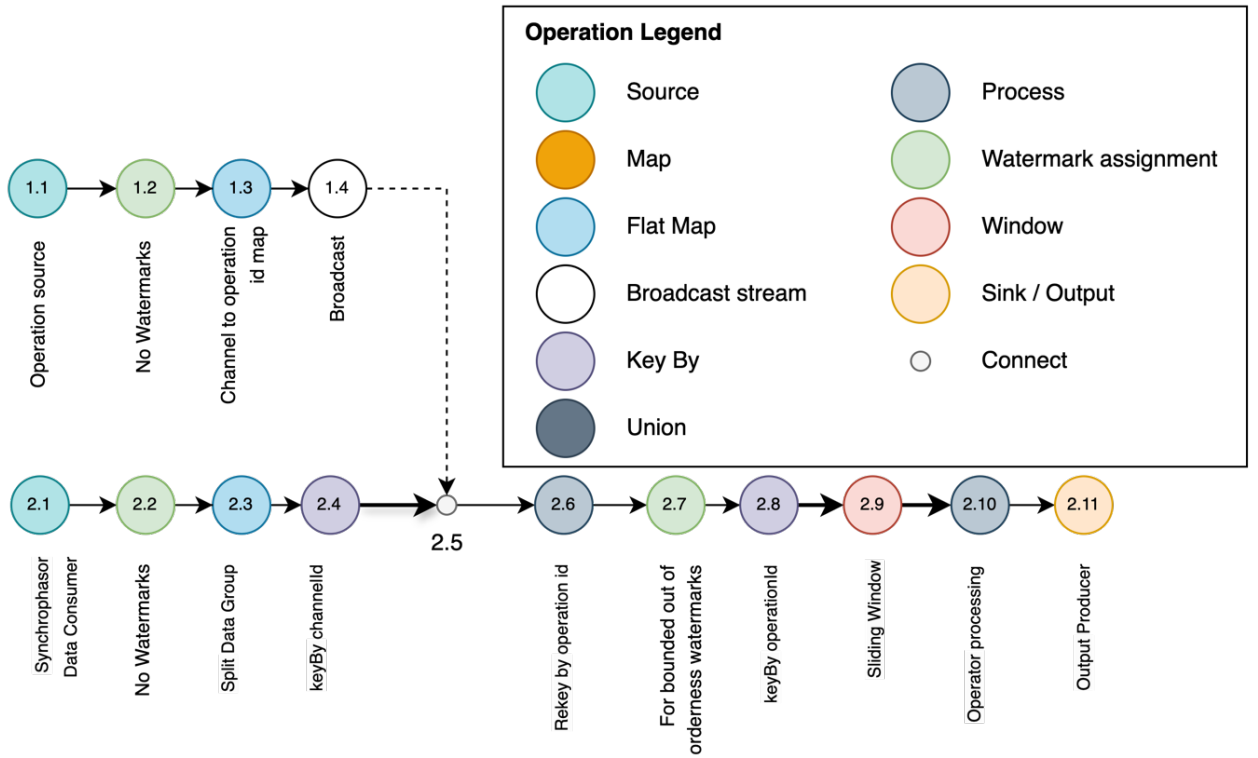
## Detailed data transformations

- $t_i$ $C_m$ Represents a measurement at time $t_i$ with channel ID $C_m$
- $O_n$ $C_m$ Represents the relation between an operation with ID $O_n$ and a channel by it´s channel ID $C_m$.



# Processing Pipeline

## Dataflow model

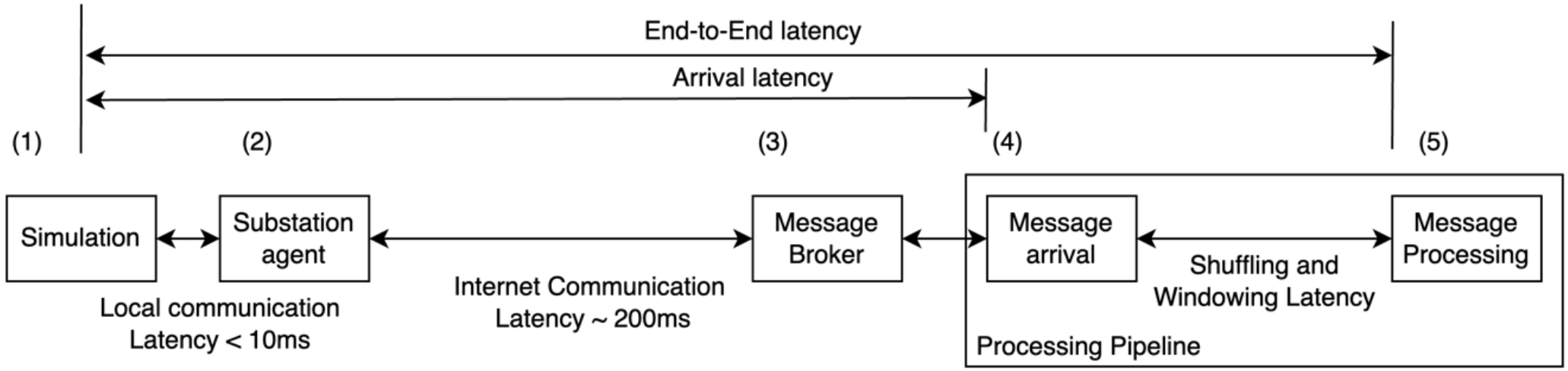


# Processing Pipeline

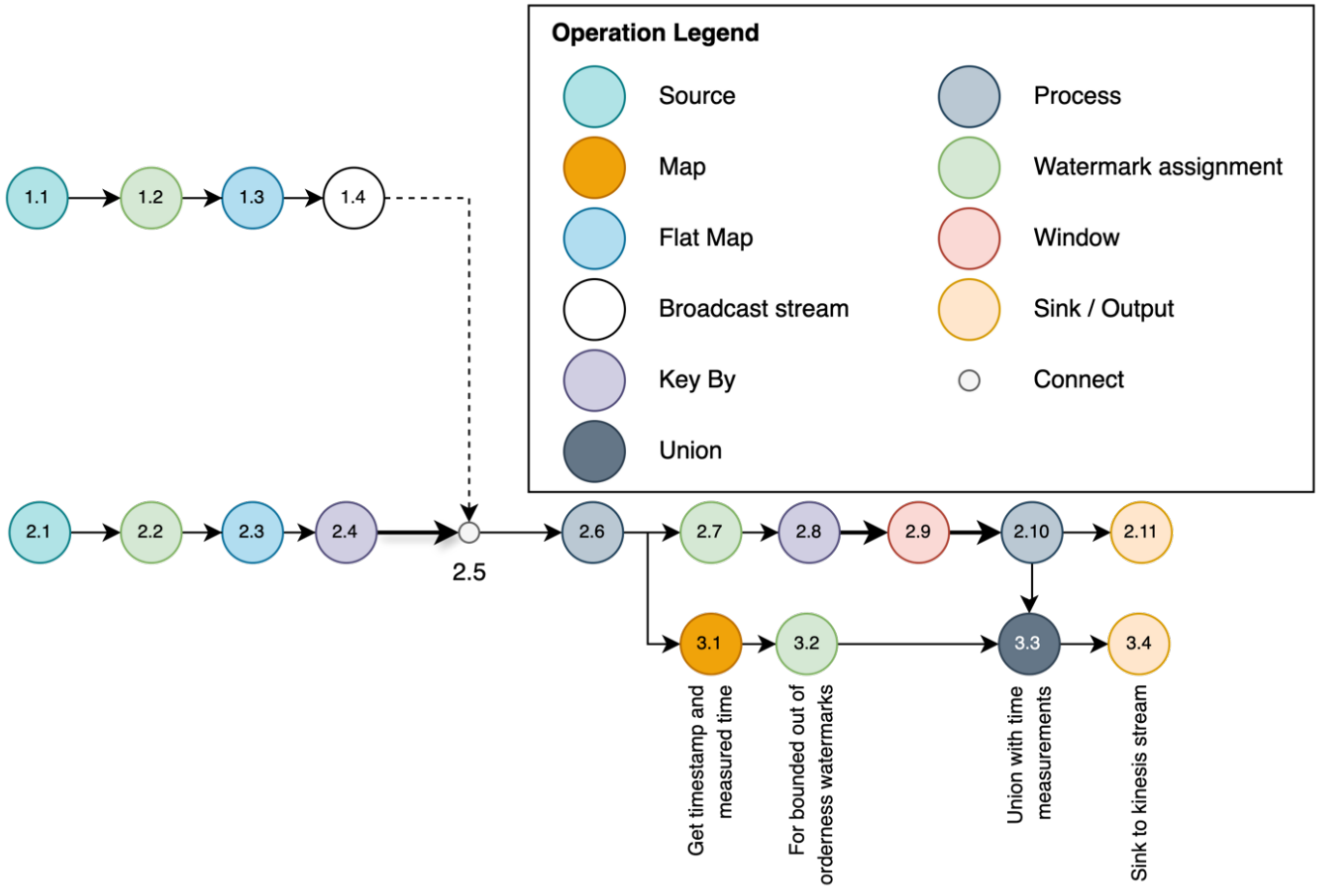
## Operation definition example

```java
public class PhaseDifferenceOperator extends AbstractOperation {
    @Override
    public void process(Map<Long, WindowBuffer<? extends QuantityEvent>> windows) {

        long VA_RE_A = operationDefinition.getBaseChannelIds().get(0);
        long VA_IM_A = operationDefinition.getBaseChannelIds().get(1);

        long VA_RE_B = operationDefinition.getBaseChannelIds().get(2);
        long VA_IM_B = operationDefinition.getBaseChannelIds().get(3);

        long timestamp = windows.get(VA_RE_A).getTimestamp();

        Double va_re_a = ((RealQuantity) windows.get(VA_RE_A).stream().findFirst().get()).value;
        Double va_im_a = ((RealQuantity) windows.get(VA_IM_A).stream().findFirst().get()).value;

        Double va_re_b = ((RealQuantity) windows.get(VA_RE_B).stream().findFirst().get()).value;
        Double va_im_b = ((RealQuantity) windows.get(VA_IM_B).stream().findFirst().get()).value;

        Double angle = Math.atan2(va_im_a, va_re_a) - Math.atan2(va_im_b, va_re_b);

        RealQuantity angleOutput = new RealQuantity(getOperationId(), timestamp, angle);
        emitOutputValue(angleOutput);
    }
}
```

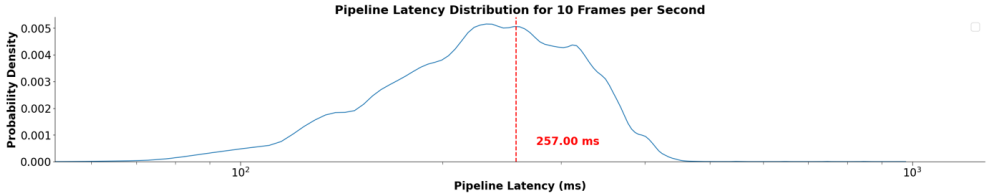
# Measurements

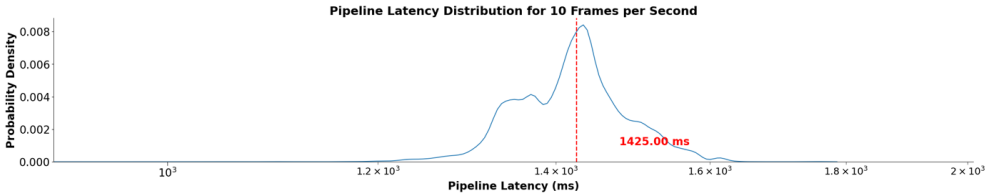


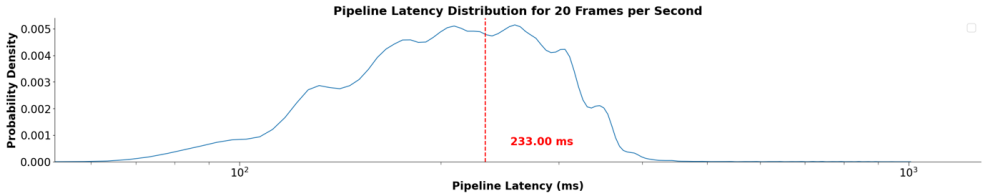
# Measurements

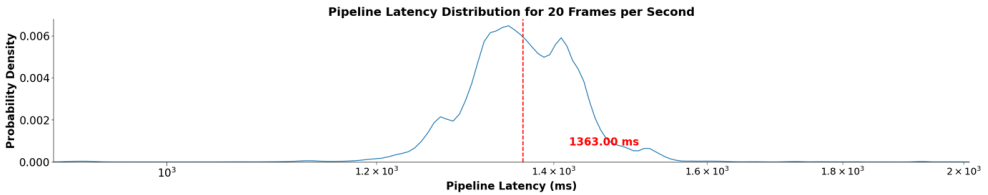


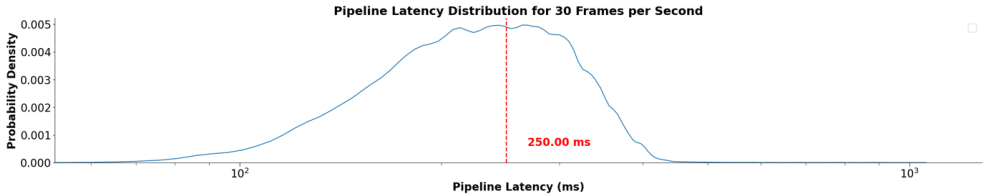
# Measurements

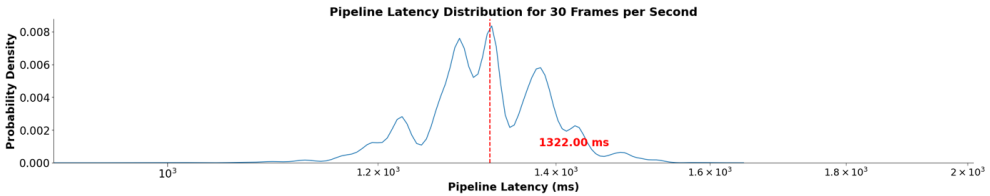


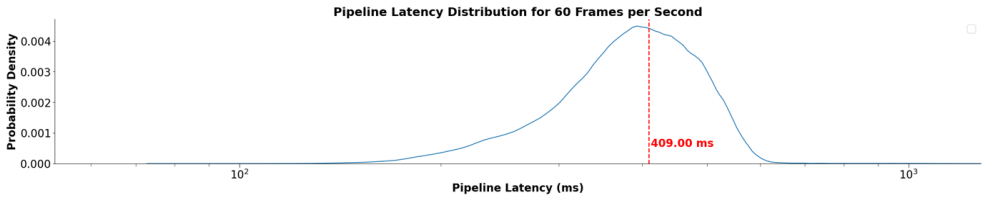



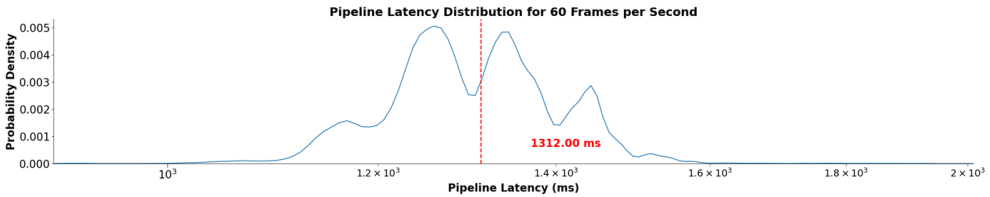



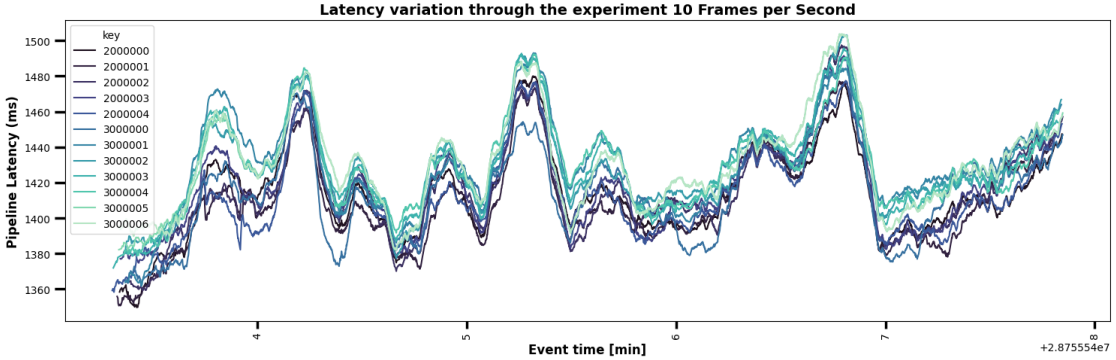



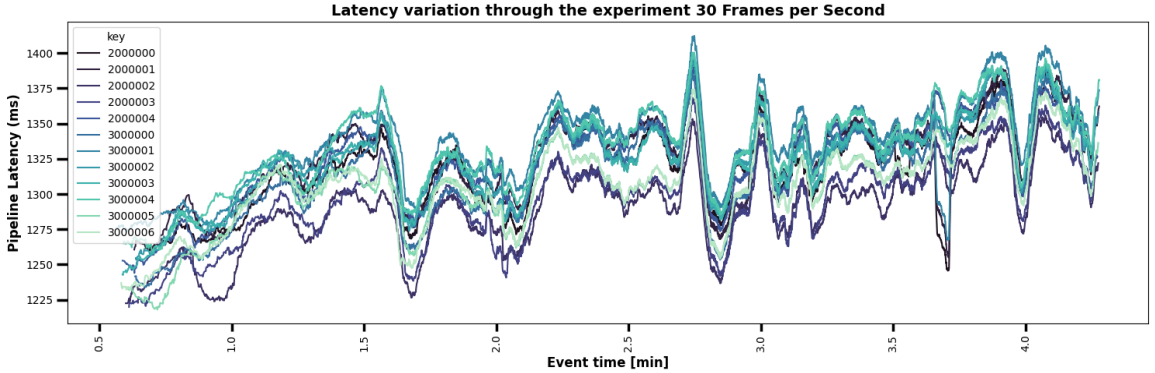



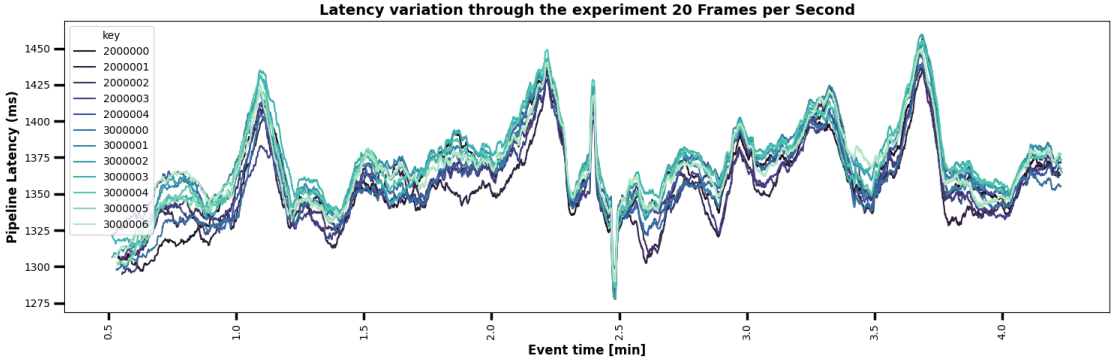



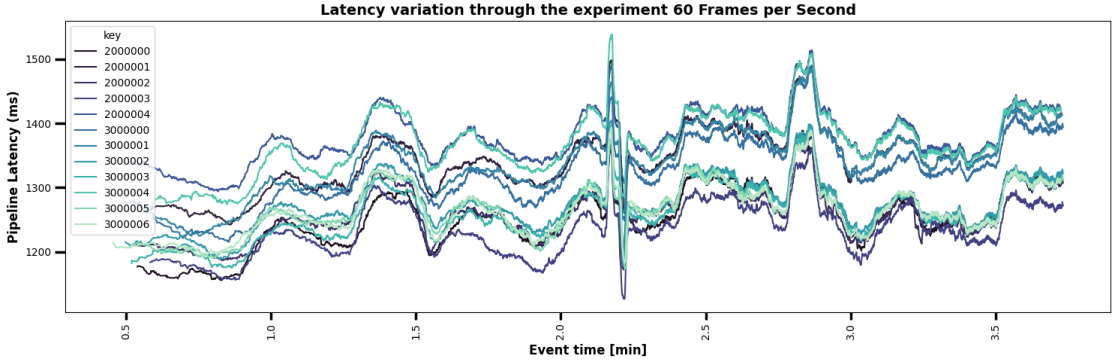



# Measurements



# Conclusions

- A detailed description of the implementation process of a cloud-based near-real-time stream processing application for synchrophasor was presented
- It was shown that even when using a state-of-the-art stream processing framework, they fall short for synchrophasor applications.
- A novel architecture for synchrophasor data processing was introduced and tested using a real-time simulator.
- An IEEE Std. C37.118™-2005 substation agent to extract synchrophasor data from substations was implemented. This was designed to overcome some of the limitations of the current approach when it comes to forwarding the data to the cloud.

# Future Work

- The proposed pipeline has multiple network shuffling steps. This limits its overall performance and points to a gap in the development of stream processing frameworks for time series and more specifically, synchrophasor data.
- The data ingestion strategy can be further optimized by implementing an IEEE Std 1344™ (1995) gateway (STTP).
- A real time simulator was used to test this system. However, testing of this at scale needs arbitrarily large amounts of data to be produced. A general framework for testing synchrophasor big data architectures has not yet been proposed.

# Thank You!