StreamBox-TZ:

Secure Stream Analytics at the Edge with TrustZone

USENIX ATC'19

2022.04.19

Background

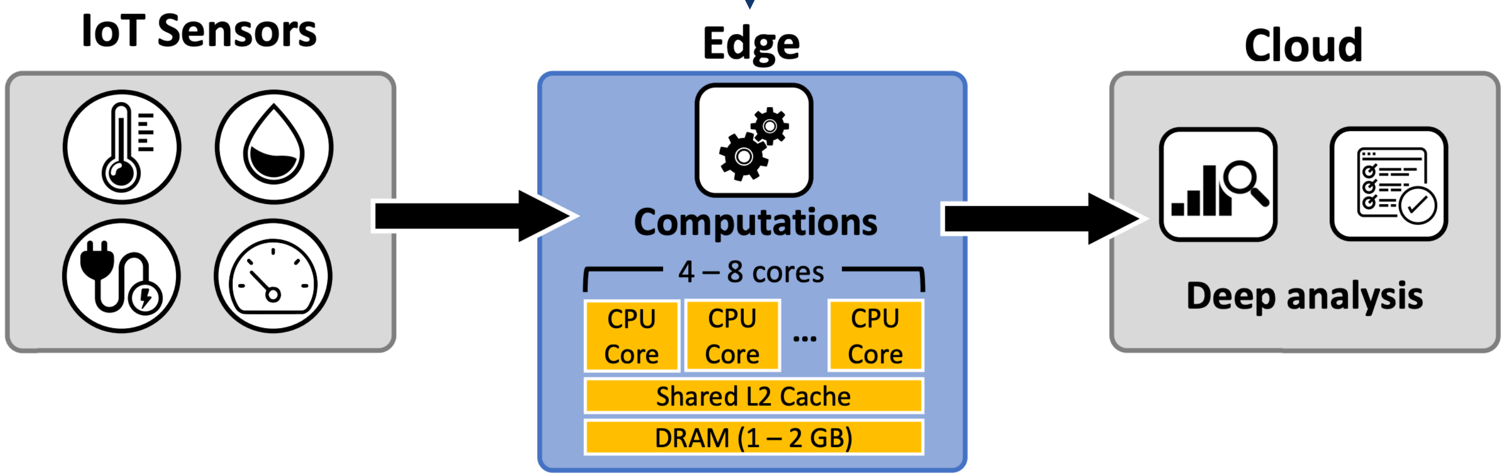
Edge Processing

The large telemetry data streams must be processed in time

But transmitting data …

- High cost
- Long delay

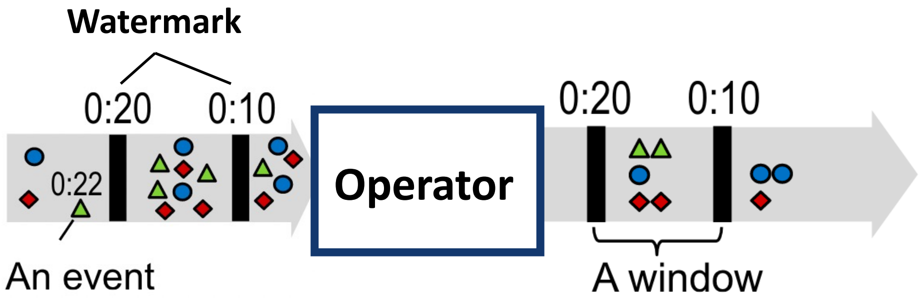
} Edge Processing



Background

Stream Analytics

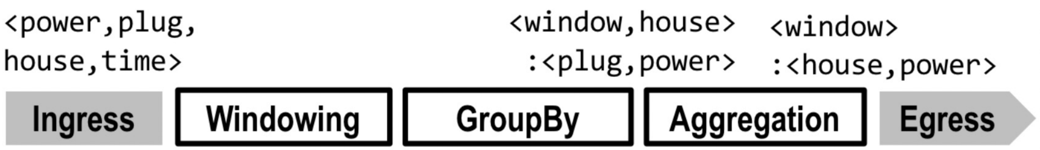
- A data stream consists of sensor **events**
- A **pipeline** may maintain its internal states organized by windows at different operators



Stream analytics engine:

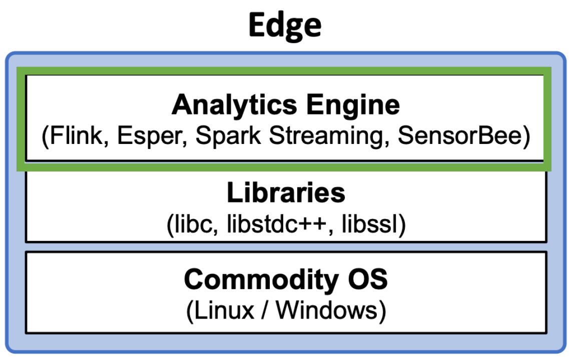
a runtime framework executes the stream pipelines

- Data functions: data move and computations
- Control functions: resource management and computation orchestration



A simple analytics pipeline

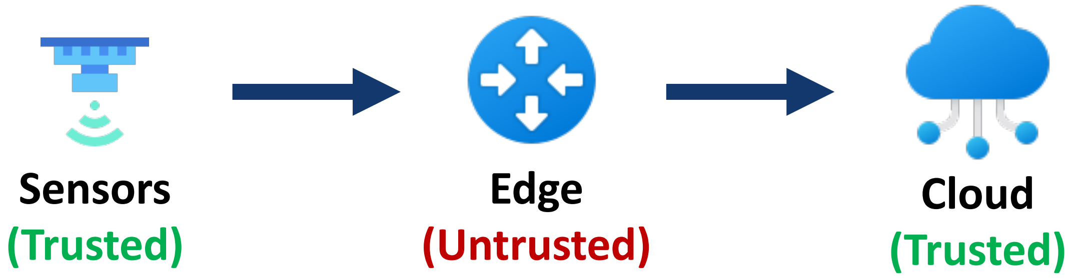
A stream analytics engine **ingests** the data at the pipeline ingress, **pushes** the data through the pipeline, and **externalizes** the results at the pipeline egress

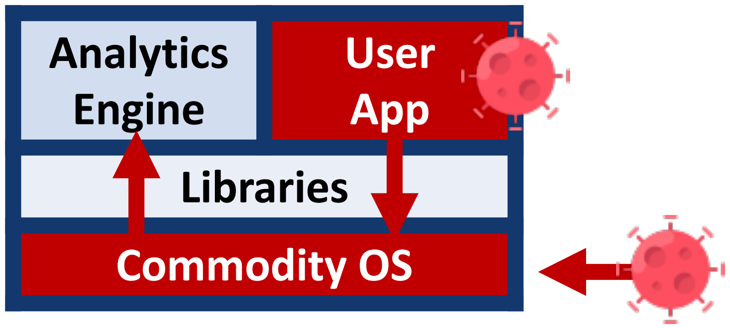


Background

Security Threats on Edge Processing

- Common IoT weakness
  a. Lack of professional supervision
  b. Weak configurations
  c. Long delays in receiving security updates
- Wide attack surface of sophisticated components on the edge
- High-value target to adversaries





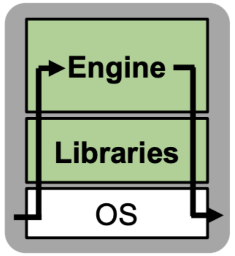
Threat Model

Goals & Challenges

Existing systems:

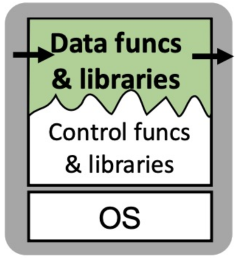
Pull **entire** user apps and libs to the TEE:

- Large and complex engine
- Potentially vulnerable Libraries



Partitioning apps to suit a TEE:

- Mismatch TEE's limited memory
- Unsuitable for existing engines

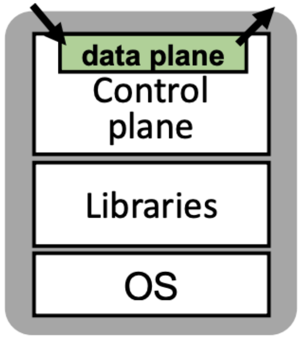


Lack support for stream analytics, key computation optimizations, or specialized memory allocation

Design goals:

For stream analytics over telemetry data on an edge platform:

- confidentiality and integrity of IoT data, raw or derived
- verifiable correctness and freshness of the analytics results
- modest security overhead and good performance



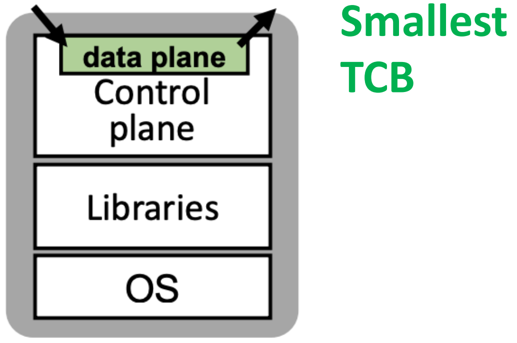
Smallest TCB

Goals & Challenges

Design goals:

For stream analytics over telemetry data on an edge platform:

- confidentiality and integrity of IoT data, raw or derived
- verifiable correctness and freshness of the analytics results
- modest security overhead and good performance



Challenges:

- **What functionalities** should be protected in TEE and behind **what interfaces**?
- **How to execute** stream analytics on a TEE's low TCB and limited physical memory while still delivering high throughput and low delay ?
- As both trusted and untrusted edge components participate in stream analytics, **how to verify** the outcome ?

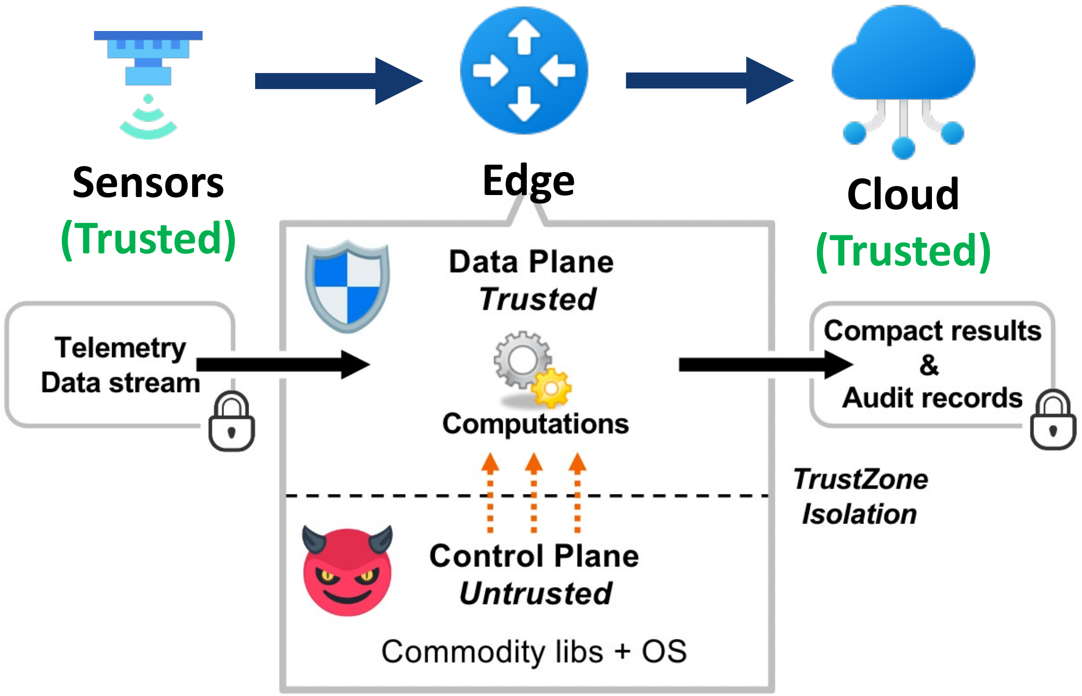
Main Idea

StreamBox-TZ (SBT)

A secure engine for analyzing telemetry data streams

Built on **ARM TrustZone** on an edge platform

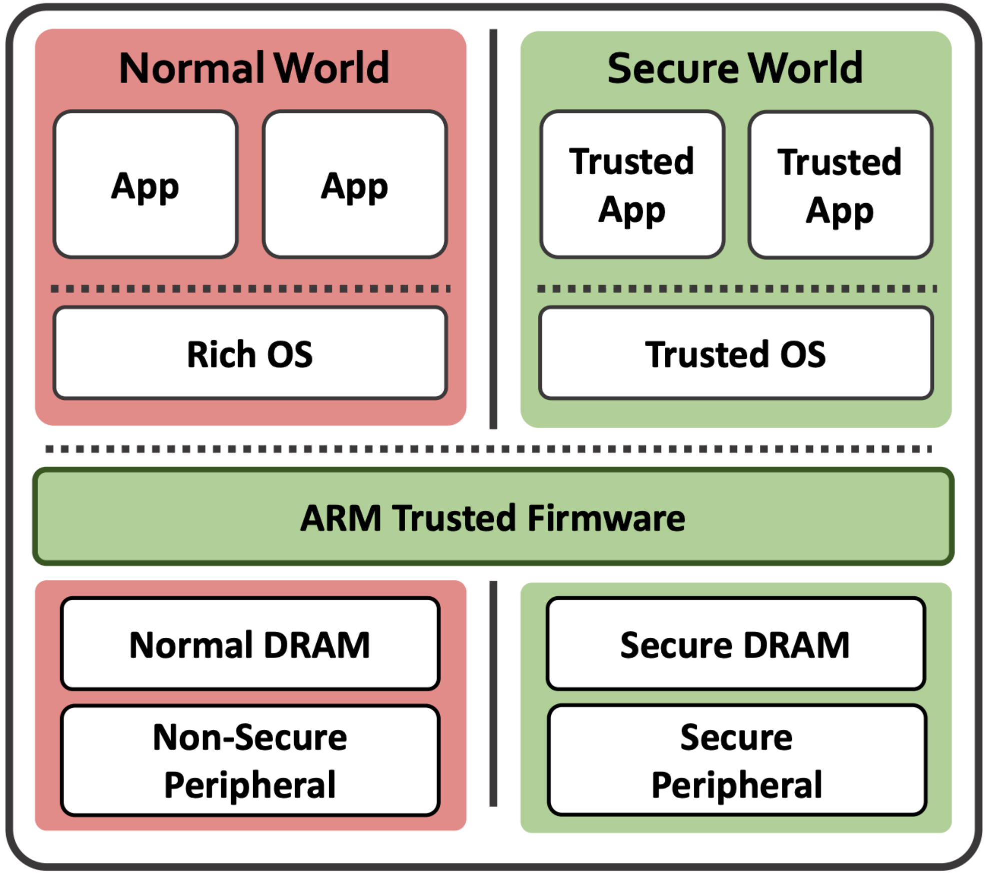
- Architecting a data plane for protection
- Optimizing data plane performance within a TEE
- Verifying edge analytics execution



Idea 1: Architecting a Data Plane for Protection

ARM TrustZone

Most modern ARM cores are equipped with TrustZone - a security extension for TEE enforcement



Features:

- Logically partition a platform’s hardware resources into a **normal world** and a **secure world**
- CPU cores independently switch between two worlds
- **Trusted IO**: Any peripheral owned by the secure world is completely enclosed in the secure world

Idea 1: Architecting a Data Plane for Protection

Data plane:

Consist of:

- Trusted primitives
- Minimum runtime functions

Generate:

- Computation results
- Audit records

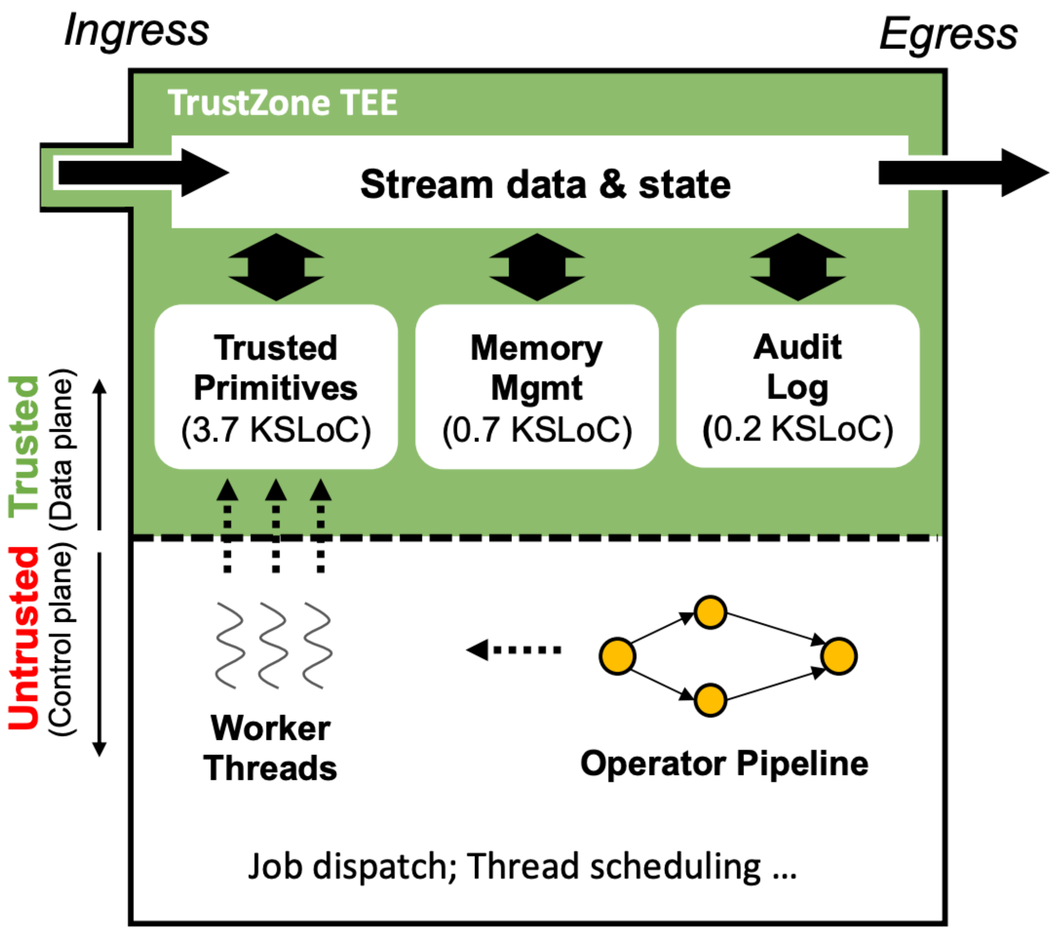
Control plane:

Control functions

- Orchestrate the execution of analytics pipelines
- Create plentiful parallelism among and within operators

Interface between data plane & control plane

- Narrow, shared nothing
- Only 4 entry functions

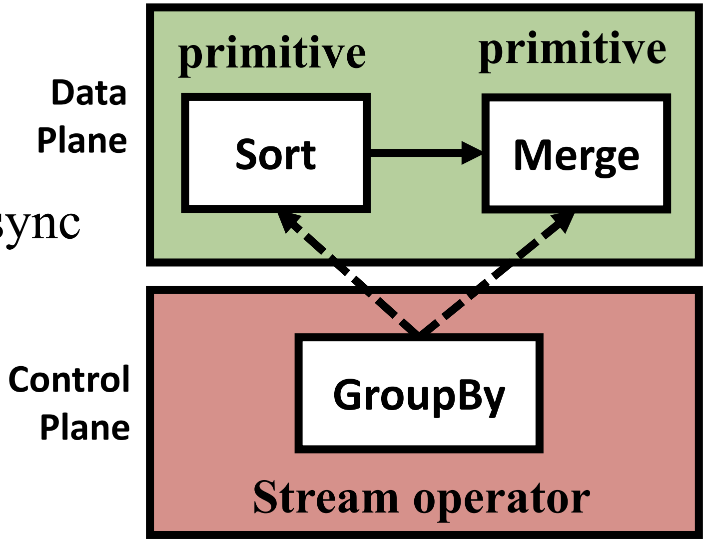


Idea 2: Optimizing data plane performance within a TEE

Parallel Execution - Trusted Primitives

Features:

- Building stream operators
- Stateless, single-threaded, oblivious to sync



Trusted Primitives

Sort, **Merge**, SumCnt, TopK, Concat, Join, Count, Sum, …

SortByKey , **GroupByKey**, Windowing, AvgPerKey, …

Popular Stream Operators

Optimization:

- Control plane invokes **multiple primitives** from **multiple worker threads**, which then enter the TEE to execute the primitives in parallel → **Contrast to existing secure analytics engines**
- Map data parallelism to **vector instructions** of ARM → **Better than STL in C++**

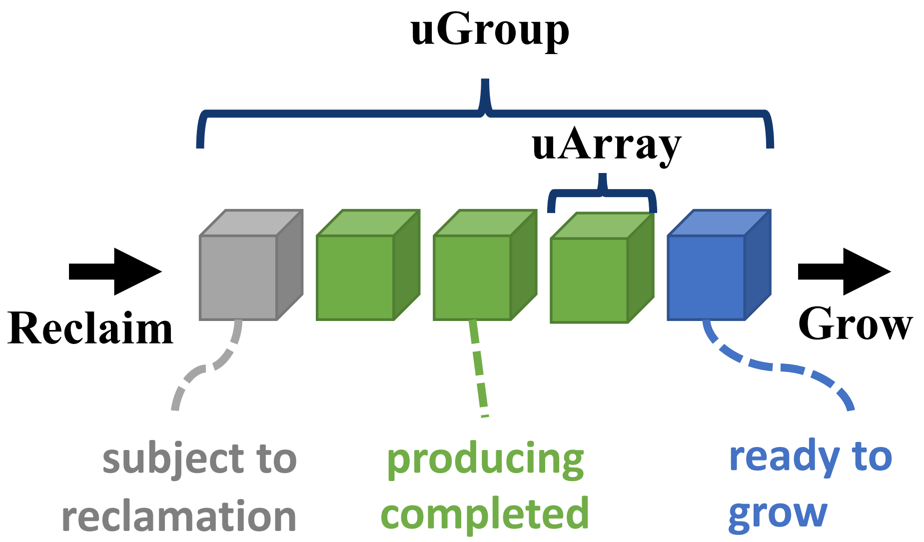
Idea 2: Optimizing data plane performance within a TEE

Memory Management - Unbounded Array (uArray)

Challenges: **(1)** Space Efficiency **(2)** Lightweight

Design of uArray:

- Encapsulate all data in a pipeline:
  - ➢ data flowing among trusted primitives
  - ➢ operator states
- Append-only buffer in a contiguous memory region
  - Growing transparently
  - Growing by updating an integer index

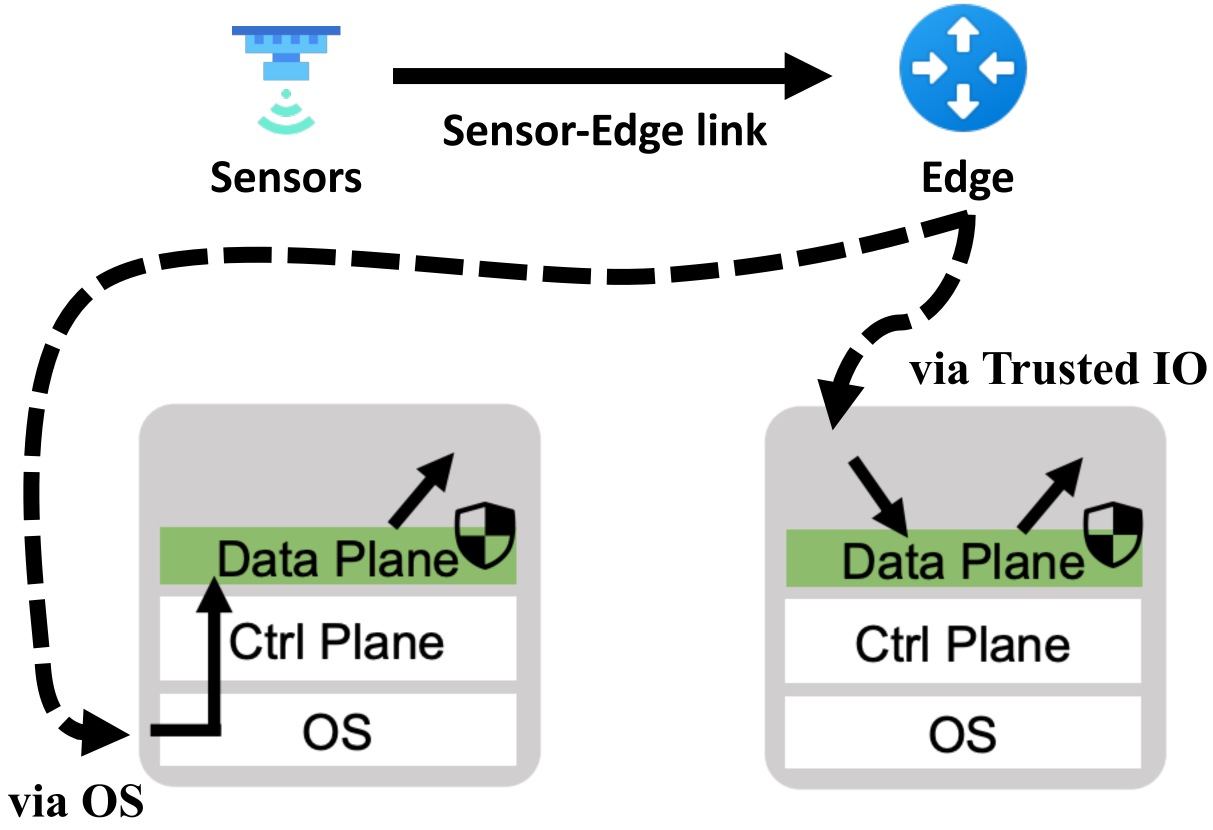


Design of uGroup:

- Co-locate multiple uArrays as a uGroup → **Reclaim consecutively**
- Place uGroups far apart by leveraging the large virtual address space dedicated to a TrustZone TEE → **Avoid collision and expensive relocation**

Idea 2: Optimizing data plane performance within a TEE

Low Data Ingestion – Trusted IO



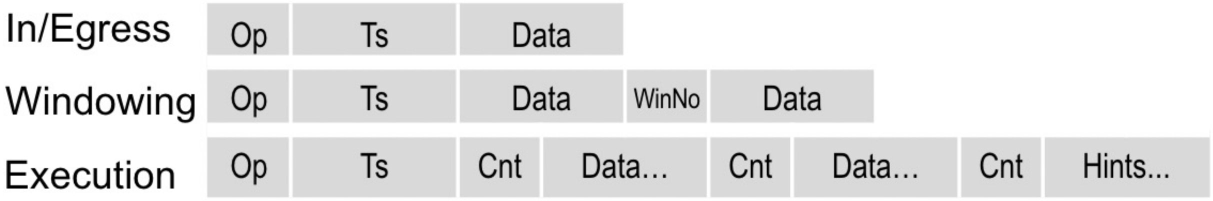
Features:

- Ingest data straightly through trusted IO without a detour through the untrusted OS
- Avoid copying and decrypting data before processing

Idea 3: Verifying edge analytics execution

Audit record:

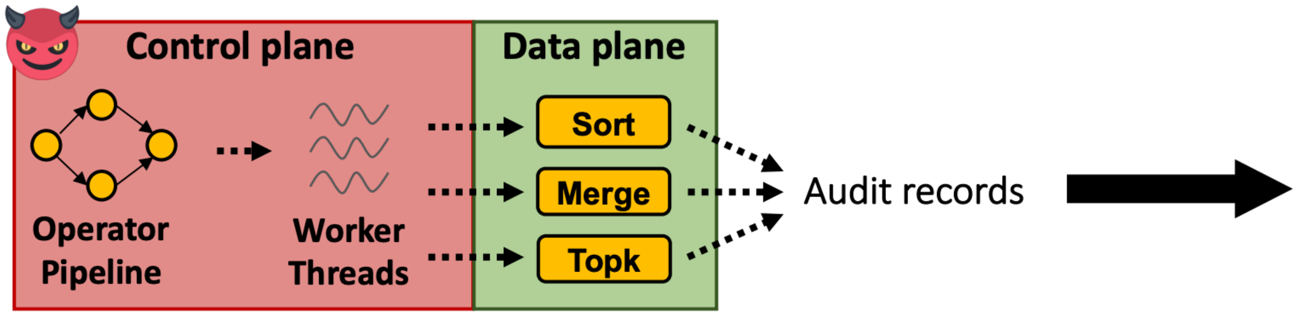
- Generated by data plane when invoked
- Monitor dataflows among primitive instances
- Compress before uploading



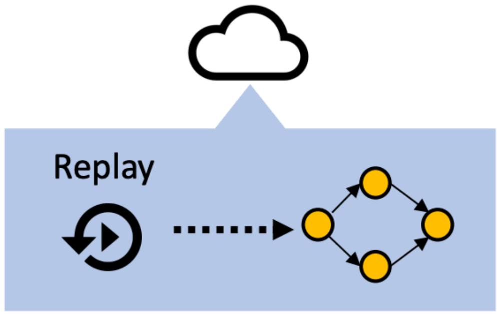
layout of audit records

Cloud verifier:

- Replays audit records
- Verify:
  - **Correctness**: all ingested data is processed correctly
  - **Freshness**: the pipeline has low output delays



Data plane generate audit records



Symbolic replay

Summary

- **What functionalities** should be protected in TEE and behind **what interfaces**?

  Architect a **data Plane for protection** and use a narrow interface to invoke

- **How to execute** stream analytics on a TEE's low TCB and limited physical memory while still delivering high throughput and low delay ?
  - **Trusted Primitives** for parallel execution
  - **Unbounded Array** for memory management
  - **Trusted IO** for low data ingestion

- As both trusted and untrusted edge components participate in stream analytics, **how to verify** the outcome ?
  - Capture coarse-grained dataflows and generate **audit records**.
  - Replays the audit records for attestation by **cloud verifier**

Evaluation

- Does SBT result in a small TCB ?

Memory Management

- 9× fewer than glibc's malloc
- 20× fewer than jemalloc

Total

richer stream operators within a
2× smaller TCB than VC3

SBT support data-intensive computation on a **minimal TCB**.

Data Plane (Trusted)

Primitives*	Mem Mgmt*	Misc*	**Total**
3.7K (32.5 KB)	0.7K (6 KB)	0.6K (4 KB)	**5K (42.5 KB)**

Control Plane (Untrusted)

Control	Data types*	Operators*	Test*	Misc*	**Total**
23K	1.3K	4.1K	1K	1K	**31K (348 KB)**

Major Libraries (Untrusted)

glibc 2.19	libstdc++ 3.4.2	libzmq 2.2	boost 1.54	**Total**
1135K	110K	13K	37K	**1.3M (3.1 MB)**

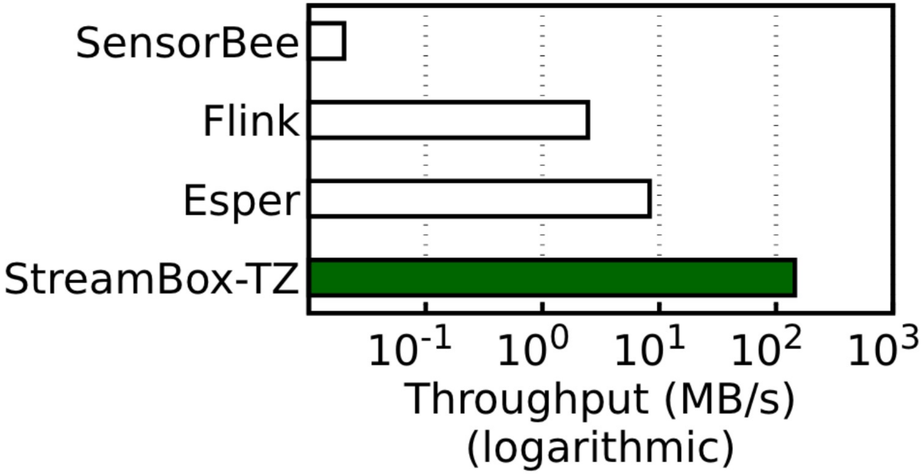
* New implementations of this work. Total = 12.4K SLoC.

Evaluation

- What is SBT's performance and how is it compared to other engines ?

  - *SensorBee*: designed for sensor data processing on a single device
  - *Esper*: designed for a single machine
  - *Flink*: designed for distributed environment and known for good single-node performance

Benchmark: Windowed Aggregation (WinSum)



StreamBox-TZ achieves **much higher throughput** than commodity insecure engines on HiKey.

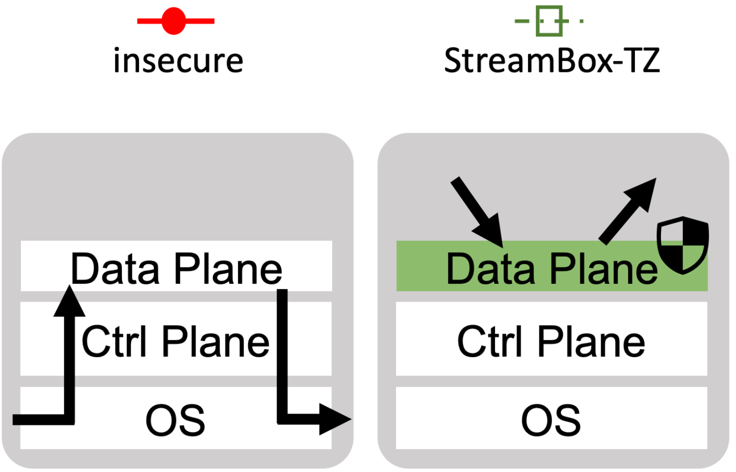
Evaluation

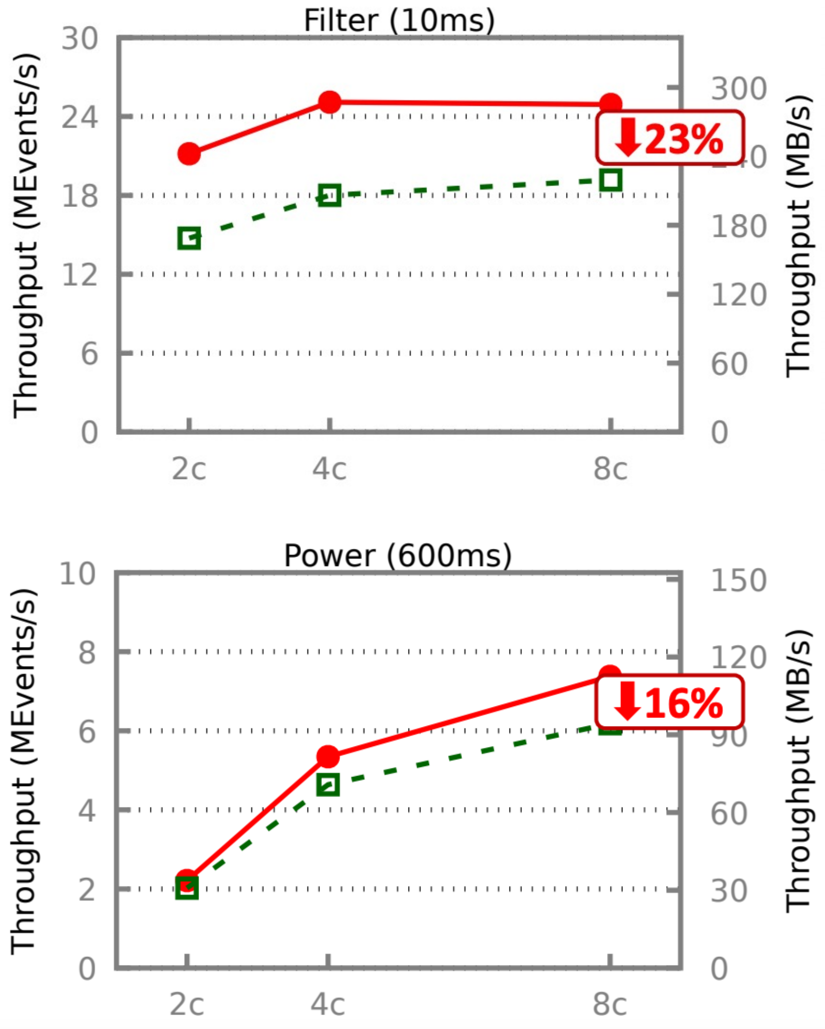
- What is the overhead ?

Benchmark:

➢ Filter: filter out input data, of which field falls into to a given range in each window

➢ Power: find out houses with most high-power plugs

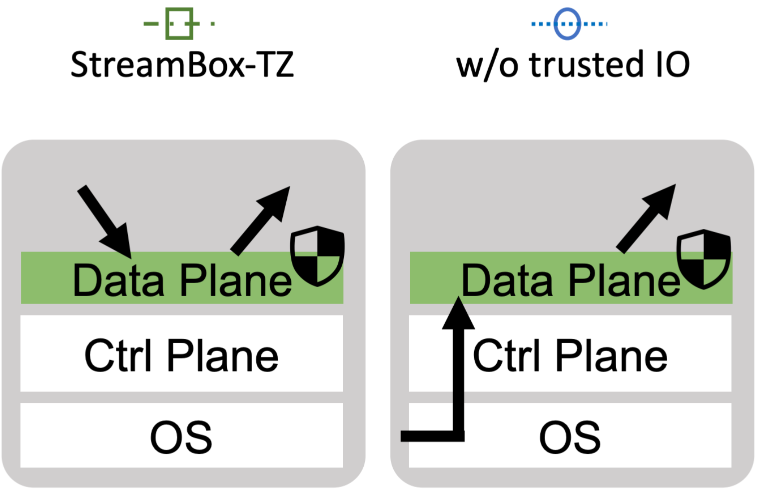




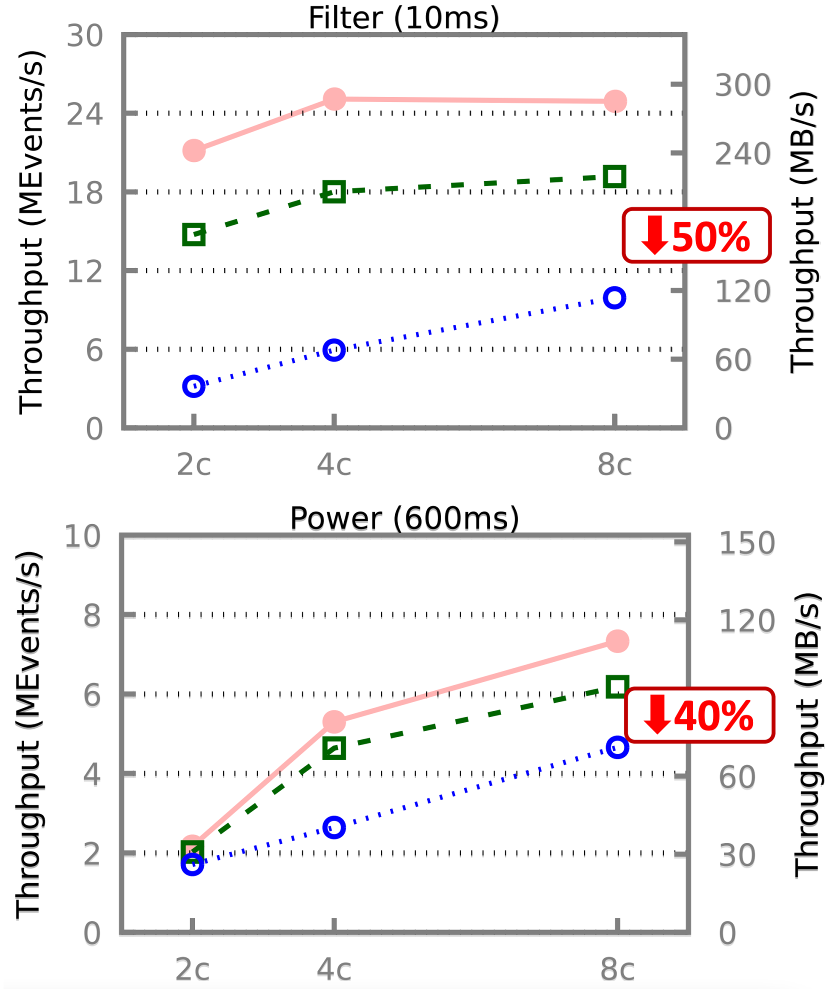
SBT only imposes **modest security overhead**

Evaluation

- How do our key designs impact performance ?

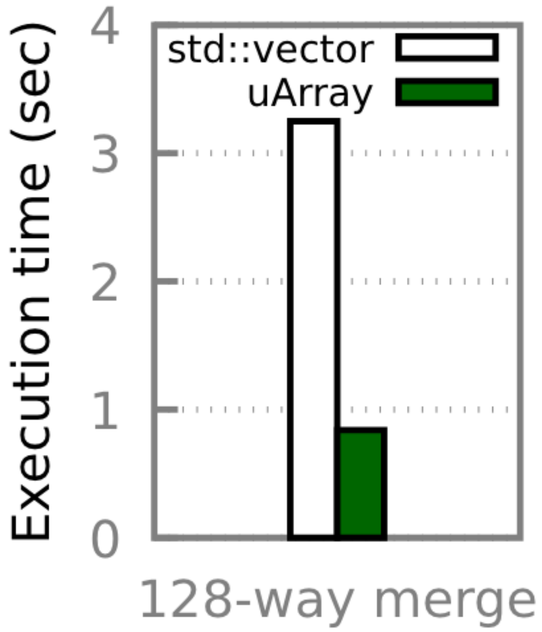


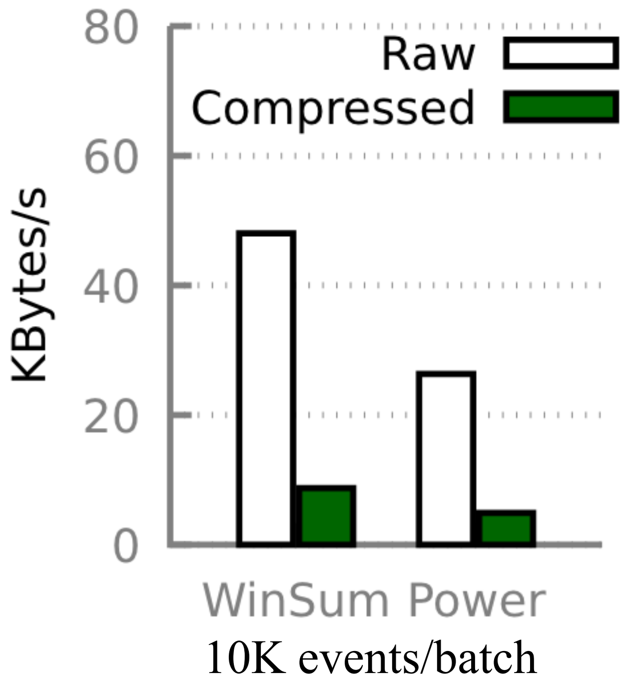
SBT outperforms the SBT w/o Trusted IO by **up to 50% in throughput** due to reduction in moving ingested data

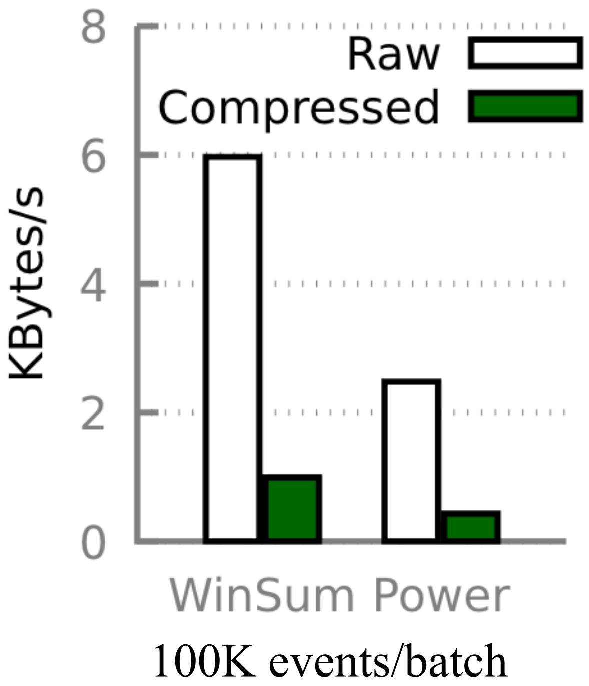


Evaluation

- How do our key designs impact performance ?







On-demand growth of uArrays is **4× faster** than std::vector

Compression of audit records **saves uplink bandwidth** substantially