Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Live Adaptation of Privacy-Enhancing Technologies in Connected Vehicles' Data Pipelines

Xinxin Zhu

Course of Study:	Autonome Systeme
Examiner:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Supervisor:	Yunxuan Li, M.Sc. Dr. rer. nat. Christoph Stach
Commenced:	April 26th, 2023
Completed:	October 26th, 2023

Abstract

Connected vehicles are able to acquire and share enormous amount of information with various onboard sensors and network communication. The information sharing can cause privacy concerns. The uniqueness of these concerns lies in the mobility of connected vehicles, which encounter different situations regularly. Different situations may have different requirements on the intensity of privacy protection. Thus, a situational policy for privacy protection should be applied. To enable a connected vehicle to carry out the situational policy correctly and effectively, we propose "Adaptive Privacy in Flow", a framework that utilizes stream processing technology to adapt the privacy-enhancing technologies within a connected vehicle's data pipeline through continuous evaluation of the environments. Our framework provides a solution of applying the situational privacy policy accordingly to the sensor data. It reacts to situational changes automatically, without restart or user intervention. Besides, it allows simultaneous and individual dynamic privacy protection for different third-party applications. Moreover, it is capable of handling the diversity of the data types in connected vehicles, from simple scalar value to complex data type like the images, ensuring a comprehensive privacy protection.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

Acronyms

CAN Controller Area Network. 20

CB Controlled Buffer. 64

CEP Complex Event Processing. 50

CV Connected Vehicle. 17

DAG Directed Acyclic Graph. 21

DBMS Database Management System. 21

DP Differential Privacy. 22

ECU Electronic Control Unit. 20

EV Evaluation Latency. 62

FIFO First-In-First-Out. 36

FP Feedback Processor. 64

GPS Global Positioning System. 17

HE Homomorphic Encryption. 22

HPC High-Performance Computing Platform. 27

IMU Inertial Measurement Unit. 20

IoT Internet of things. 83

IS Instantiation Latency. 62

KA Key Assigner. 64

LBS Location-Based Services. 83

OR Order Restore. 64

PE PET Enforcement. 64

PET Privacy-Enhancing Technology. 17

PR Processing Latency. 62

PS Presentation Latency. 62

SB State Building Latency. 63

SC Synchronization Latency. 62

SD Switching Decision. 64

SE Situation Evaluator. 64

SL Selector. 64

SMC Secure Multiparty Computations. 22

SPS Stream Processing System. 21

SW Switching Latency. 62

TD Triggering Data. 64

TR Transmission Latency. 62

TTL Time to Live. 40

1 Introduction

With the help of various on-board sensing equipments, the Connected Vehicles (CVs) gather, share, process, compute and release information regarding themselves and the environment [LCZ+14]. On the one hand, sharing the acquired data have benefits, such as for the purpose of safety. For example, Chen et al. [CTS14] has proposed *surveillance-on-the-road*, which leverages the mobility of the vehicle to track suspicious vehicles on the road. This work utilizes the onboard camera and vehicular network to perform surveillance. On the other hand, the gathering and sharing of information poses privacy concerns. For example, the conversation of the driver and occupants can be recorded at all times by the on-board voice assistants, since it waits for specific keyword for being activated. Furthermore, the voice will be sent to the backend for further processing [SGB+22]. Another example is the contribution of Jen Caltrider [Jen23]. According to this article, modern automobiles are capable of gathering in large amount details about the user, including the driving behavior and even medical data. Such information is used to infer the interests of the user. From these examples, it's clear to see that a solution is needed to protect the privacy of the user while ensuring the usability of the data, so that we can still benefit from the shared data. Such solution is the Privacy-Enhancing Technology (PET), which prevents unnecessary or unwanted processing of personal data while preserves the functionality of the information [HB95].

One specialty in the privacy concerns brought by CVs is the mobility nature, as is already leveraged in the example of Chen et al. [CTS14]. The CVs encounter different situations while moving around. In these situations, the necessity and the strength of privacy preservation may differ significantly. The CVs should adjust its privacy protection policy to meet those requirements. This can be illustrated through the following examples:

E1 *Notifying the fleet management after an accident or breakdown*: For most of the journeys, like commuting from home to the workplace, an employ can hide the route and driving behavior from the company's fleet management by sending perturbed data. However, if the company car breaks down or involves in an accident, the fleet management should be notified as soon as possible with the exact location of the car. Additional driving data may also be provided to the fleet management in order to investigate the reason of the accident or breakdown. If the location and the data relating to driving behavior are still hidden, the perturbed data loses its value for analysis.

E2 *Crossing an international border*: Regulations and data protection legislation may differ between the nations that share a border. Similar to the maps in a navigation application, the regulations could be either preloaded before departure or downloaded during the drive. A CV that travels from home country to a foreign one can prepare the required privacy policies beforehand, or receives the regulations on-the-fly as it approaches the border. The receipt and application of the foreign country's regulation is based on the CV's Global Positioning System (GPS) data.

From these examples two types of situational changes can be identified. In E1 the situational change originates from the changes in the vehicle's own state. The situational changes in E2 is resulted from the environment.

The examples above illustrate the fact that the appropriate privacy policies under different situations can be different. A setting of PETs without considering the situation cannot address the issue of dynamic environments. To handle the dynamics, PETs with different intensities should be switched against each other. Moreover, it is important in the scenario of CVs that the switching procedure should be completed automatically. Unlike privacy protection in web brower, where the user are notified to give privacy preferences, the driver shouldn't be distracted by any means in moving vehicles. Thus, the predefined privacy policy should be comprehensively defined, and the privacy protection should follow the predefined privacy policy without the intervention of the user. In addition to automatic response, the switching action should also be precise. The perceived data should be imposed with the correct PET, so that possible privacy breaches can be avoided. A solution of PET enforcement, which is responsive and adaptive to the changing situations, is needed. This is the core of this work.

In the previous work of Li et al. [LHSM23], a general privacy protection framework for CVs is established. However, the pipeline supporting live adaptation of PETs remains an open topic. As enrichment of the previous work, this thesis proposes *AdaPrivFlow*, Adaptive Privacy in Flow, a framework realizing live adaptation of PETs through utilizing the data stream processing technology. *AdaPrivFlow* establishes a data pipeline to evaluate the current situation continuously and apply the modularized PET in alignment to the given privacy policy.

This remainder of this thesis is as follows: Chapter 2 introduces the related background knowledge briefly, including the vehicular data and its processing with stream processing technology, as well as an overview of the related privacy-enhancing technologies. In Chapter 3, the requirements of *AdaPrivFlow* are proposed. Its architectural and functional designs for fulfillment of the requirements are presented. Following this, a proof of concept of *AdaPrivFlow* is illustrated in Chapter 4. In this chapter, the features of potential underlying technologies are analyzed. Based on the analysis, the technologies for implementation are selected. In Chapter 5, The effectiveness of the measures in *AdaPrivFlow* proposed in Chapter 3 will be evaluated. Other necessary implementation aspects are also introduced. In Chapter 6 the related works are reviewed. At last, the conclusion and outlook will be provided in Chapter 7.

2 Background

As is mentioned in Chapter 1, *AdaPrivFlow* leverages stream processing technology to impose the PETs on vehicle data in a CV with adaptation regarding the situation. Before the details of *AdaPrivFlow* are illustrated, we give a brief introduction to the background knowledge about the vehicle data, its processing mode and the related PETs.

2.1 Vehicle Data and Its Processing

Since *AdaPrivFlow* processes the vehicle data, the common types of data in a CV should be visited. Their properties should be investigated. In this way, *AdaPrivFlow* can be tactically elaborated to handle the challenges brought by the properties.

Rettore et al. [RMVL19] proposed the concept *vehicle data space*. In this concept, the data in the modern vehicles originates not only from the embedded sensors, but also from the environment. The embedded sensors belong to *intra-vehicular* sensors. They stand for all sensors that monitor a vehicle's state, the driver's actions, or the environment conditions around the vehicle. The information sources in the environment are referred to as *extra-vehicular* sensors. Rettore et al. [RMVL19] provided a pictoral presentation of the categorization, which is shown in Fig. 2.1. It could be seen from the figure that the intra-vehicular sensors not only senses the vehicle's own

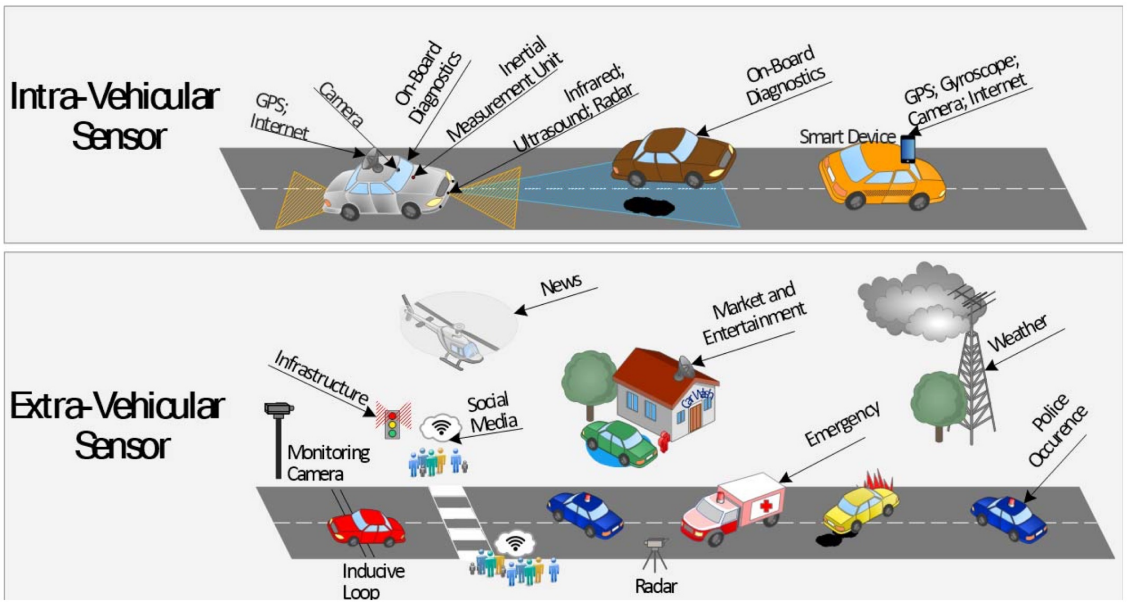


Figure 2.1: Vehicular Data [RMVL19]

state, but also perceives the environment. Typical sensors that obtain the vehicle's state are GPS, Inertial Measurement Unit (IMU). The camera and radar are typical sensors that perceives the environment. Extra-vehicular sensors, on the other hand, can be either Internet feeds, such as the weather information, or broadcast information from other entities, such as the emergency signal from the ambulance and the signal of police occurrence.

Since *AdaPrivFlow* focus on the vehicle data generated by the onboard sensors, in the following, we focus only on the *intra-vehicular* sensors. As is shown in Figure 2.1, the common data from onboard sensors in a CV have a large spectrum of origins. Due to this reason, the vehicular data is featured with diversity. Beside scalar values, there exists other types of sensor reading, such as camera, LiDAR and radar. These data types have varying properties, for example:

- *Data Volume*: Sensor readings, such as temperature and engine speed, are essentially a float number. Depending on the precision, the size is usually 4 Byte or 8 Byte. Sensor readings from perceptional devices, such as image data or LiDAR data, has much larger volume [Pim17]. For example, the image data is a two-dimensional array pixels. A typical image data have the size in the magnitude of Megabytes.
- *Data Refresh Rate*: The sampling rate of data from the Electronic Control Unit (ECU) alone may vary depending on the type of the engine. According to Vector Informatik GmbH [Veca], for internal combustion engines, the rate is a few kHz, while the value for electric motors falls in the range of MHz. The sampling rate of sensor with purpose of autonomous driving is much lower, compared with sensors in the motors. The sampling rate of a typical camera is 30 Hz, and of a typical radar sensor, 15Hz [Pim17].

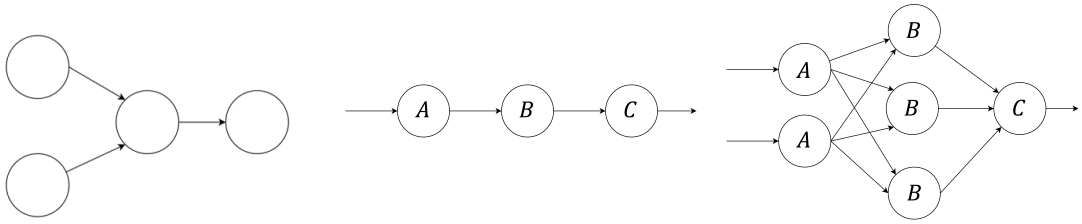
The vehicular data provide comprehensive information of the vehicle. By processing and fusion of (a subset of) the data, the *vehicular context* is perceived [TVC+15]. In this way, situations can be detected. In the work of Terroso-Sáenz et al. [TVC+15], the context is categorized into dynamic context and static context. Static context includes the information about the owner and the feature of the vehicle, such as the feature of the engine, the interior and the body. Dynamic context contains the location, the time, the activity and the identity of the occupants. Typical information contained in the activity are the speed and the itinerary. The dynamic context lies in the interest of this work, since it can be composed by sensor readings. The *situations* in this work is then a subset of the dynamic context.

The collection of sensor data for processing and sharing of the data rely on the communication architecture, which evolves from Controller Area Network (CAN) towards Automotive Ethernet with the development of the technologies [HMVV13]. Thus, the sensors onboard form a sensor network. According to Gama and Gaber [GG07], data stream processing is an adequate method to handle the data from the sensor network. In the following, the focus will shift to the data stream processing technology.

Data stream is formally defined by Stephens [Ste97] as an infinite list of elements $a_0, a_1, a_2 ...$ taken from a certain data set R. The stream is formalized mathematically as a function

$$a : T \rightarrow R$$

where $T = \mathbb{N} = \{0, 1, 2, ...\}$ represents discrete time. Data stream processing has several characteristics:



(a) Example of Stream Processing Application

(b) User graph

(c) Execution graph

Figure 2.2: Directed Acyclic Graph Representation of Stream Processing Applications

- *Distributed by nature*. In general, a stream processing system consists of numerous heterogeneous computing devices. For example, the data source is a mobile device. The data is shared via network and analyzed in the cloud. What's more, it's a common practice to use a large number of servers to solve the issue of time-varying load spikes [CBB+03]. These scenarios have the nature of distributed computing.
- *Active processing*. Traditional Database Management Systems (DBMSs) works in a passive manner. The processing of data is initialized by a user query. To the best case period queries are running over the stored data in DBMSs [IAM+19]. On the contrary to DBMSs, the data in the stream is read and processed upon arrival. They are not stored before being processed [SÇZ05]. A certain amount of memory is optionally consumed to store some subset of the past items. However, no arbitrary access to past events is supported [GKMS01]. In contrast to traditional DBMSs, stream processing is event-driven or data-driven, saving the overhead of polling the results through the clients [SÇZ05].

According to Stephens [Ste97], a Stream Processing System (SPS) consists of a collection of parallelly running processes, which transmit data via channels. The processes are categorized into:

- *Source*. The source is a component that feeds data into the SPS.
- *Filter*. The filter performs atomic computation on the data. In popular streaming frameworks, they may also be called by other names, for instance, the name "operator" is used in Apache Flink.
- *Sink*. The sink passes information from the SPS to the outside.

According to Isah et al. [IAM+19], a streaming application in the common stream processing engines is represented as a Directed Acyclic Graph (DAG). Figure 2.2a shows the visualization of an exemplary stream processing application, where 2 sources, 1 filter, and 1 sink can be seen. The research of Kamburugamuve and Fox [KF16] shows that the modern stream processing frameworks distinguish "user graph" and "execution graph" in the practical point of view. The programmer implements the user graph by defining the sources, the filters and the sinks. The user graph is then compiled to the execution graph by the stream processing framework. Depending on the user definition, or on the framework, the nodes in the DAG are to be distributed to multiple computing devices [KF16]. The distributed nature of data stream processing mentioned earlier in this section

is represented in this way. An exemplary user graph is shown in Figure 2.2b and its corresponding execution graph in Figure 2.2c. The user graph suggests 3 filters. After compilation, 2 parallel instances of A, 3 parallel instances of B and 1 instance of C participates in pipeline.

2.2 Privacy-Enhancing Technologies for Connected Vehicles

As is described in Chapter 1, PETs are essential in providing privacy protection under different circumstances. In this section, a short introduction about PETs is given. Note that the theme of PETs is very complex. We only provide an overview and cover the categorization of PETs from the literature and with own ideas, which suits the scenarios in this work. Detailed discussion about the PETs is not included.

Hes and Borking are credited for creating the concept of PET in 1995 [HB95]. According to them, by limiting the processing of personal data, the PET can be utilized to protect the sensible data from being misused. Garrido et al. [GSH+21] has provided an overview of important PET in the domain of automotives. In this work, the author analyzed possible PET for several typical uses cases. The PET are categorized into:

- *Enhancing anonymity*. The PETs in this category utilized the properties of Differential Privacy (DP) and k-anonymity. DP "addresses the paradox of learning nothing about an individual while learning useful information about a population." [DR14] Random noise are introduced in the results to protect the data from malicious participants [DKM+06]. k-anonymity is achieved when each individual's information in a data set cannot be discriminated from the information of at least $k-1$ other persons [SS98]. The exemplary scenario are services based on geological information and data analytics of a group.
- *Enhancing confidentiality*. The PETs belonging to this category rely on cryptography [GSU+22]. The exemplary scenarios are information sharing and processing across organizations [GSH+21]. Homomorphic Encryption (HE) and Secure Multiparty Computations (SMC) are typical technologies. Selected operations on encrypted data can be carried out by HE as if the data were not encrypted. The output following the computation can be decrypted by the entities that holds the corresponding secret key [WK15]. SMC enables many people to jointly calculate a function over their inputs while maintaining the privacy of the inputs [CDN15].

The categorization about is enriched with the work of [Tİ+23], where the PETs are explored from the aspect of data mining. Besides the cryptographic-based methods, such as SMC, and heuristic methods, such as the ones utilizing the property of k-anonymity, a new classification of restructuring-based technology is proposed. Typical PETs belonging to this category is randomization and perturbation.

Another categorization was proposed by Kaaniche et al. [KLB20]. The PETs are classified by their deployment layers:

- *User-side techniques*. The techniques are applied in the end-user side.
- *Server-side techniques*. The techniques are applied in the servers that anonymizes database for data exchange or performing computations over encrypted data at the request of users. DP and HE are representational technologies in this category.

Table 2.1: Typical Prevalent PETs and their Features[GSH+21][KLB20][Tİ+23]

	Properties			Layers		
	Anonymity	Confidentiality	Restructure	User	Server	Channel
DP	×				×	
k-anonymity	×				×	
End-to-End Encryption		×				×
SMC		×		×		
HE		×			×	
Randonmization			×	×	×	
Perturbation			×	×	×	

- *Channel-side techniques*. Channel-side techniques include secure communications, which stands in alignment with the result from [GSU+22].

The categorization in the aforementioned literatures are summarized with typical PETs in Table 2.1. Besides the discussed attributes of the listed PETs, we propose a new categorization based on the PET's reliance on existing data. Stach et al. [SGB+22] has summarized that several approaches of data sharing while ensuring privacy correspond to operations performed on datasets. The data can either be minimized and condensed, or certain attributes can be removed, before being processed. In our work, to describe the reliance on existing data, we have

- *Stateful PETs*. They have reliance on existing data.
- *Stateless PETs*. They don't rely on existing data to operate.

Stateful PETs introduce new issues under our application scenario. That is, the preparation of the necessary history data when such kind of PET is about to be switched. Therefore, in addition to the features listed in Table 2.1, we further explored the categorization based on the dependency on existing data sets, such as history data records. A finer granularity of further classification lies in the origins of the existing data. They can be divided into:

- *Non-Ego-centric* PETs. The existing data has multiple origins. For example, practical DP in the domain of healthcare concerning the information of multiple patients [DE13].
- *Ego-centric* PETs. The existing data comes from itself. An example is the work of Al-Dhubhani and Cazalas [AC18], which obscure the user's location based on the degree to which its previously obscured locations are correlated.

3 Framework

In this chapter, we provide a comprehensive analysis of *AdaPrivFlow*. In Section 3.1.1, we derive the requirements for *AdaPrivFlow* to realize the live adaptation of PET in response to environmental conditions. We will further clarify the problem definition through a detailed examination of the scope, the prerequisites and the underlying assumptions. Section 3.2 focus on the conceptual design of *AdaPrivFlow*. As the first step, an overview of the structure is proposed to ensure the intended functionality. Following this, various components within this framework are mapped to the overview. A data pipeline is established. Having established the data pipeline, we focus on the approach of enabling the live adaptation of PETs, which will be illustrated in Section 3.3. The conceptualization begins with the evaluation of several adaptation approaches, before a design choice is settled. Following this, We deal with the mechanism of algorithm switching. We first explore the instantiation of PET algorithms, then we propose the first approach to perform PET switching that satisfies the requirements mentioned in Section 3.1.1. In Section 3.4, the proposed simple approach is evolved to a sophisticated solution with the consideration of the diversity of PETs and vehicle data. To this point, the refined design of architecture and the data pipeline is proposed, enriched by the insights of dealing with real-world challenges. A summary of the refined architecture is provided in Section 3.5.

3.1 Requirements and Assumptions

In this section, the requirements and assumptions as the cornerstone of the framework design are covered. We propose the requirements of realizing the live adaption of PETs in the data pipeline of a CV. The assumptions on the enabling technology, the input data and the applicable PETs will be introduced.

3.1.1 Requirements

We conclude the requirements of the framework in the aspect of deployment environment, application scenarios and quality of service.

R1 *Automatic reaction to situation changes without stop or user intervention.* In order to achieve automatic reaction, the CV should evaluate the situation continuously and perform adaptation based on an established privacy policy. To achieve this, the CV should persistently monitor both its internal conditions and the external environment. The adaptation process should occur seamlessly at the runtime. It's impractical that a CV should be restarted every time a changed condition occurs. Moreover, the adjustments are expected to be applied without user

intervention. Unlike web browsing or mobile apps, where the user are notified to adjust their privacy preference through prompt, it's not feasible to distract the driver from driving and make privacy decisions.

R2 *Support for adaptation for individual apps*. According to this requirement, each involved application should be applied with customized privacy policies. It's intuitive that for different applications the user would like to have different strength on the privacy protection. Permission management of mobile apps is a typical example. The requirement on different strength of privacy protection for different application not only originates from the user side, but is also scenario-dependent. The scenario E1 in Chapter 1 serves as a strong argument for such support. Although the fleet management should be informed immediately with the exact position of the CV, the location information should remain protected against other running applications, such as the navigation app. Otherwise, a privacy breach will happen.

R3 *Accurate PET Enforcement according to the defined privacy policy*. This requirement indicates two aspects of accuracy. Firstly, the framework is able to apply the right PETs under the right situation, as defined in the privacy policy. Secondly, each data record should be applied with the PET that it should comply to. This is especially important when the privacy protection is switched from low intensity to high intensity. Incorrect enforcement with low intensity PET under the condition requesting high intensity privacy preserving leads to privacy breach. Consider the switching procedure occurring the framework, it's crucial to perform accurate PET enforcement on each single data record during the transition.

R4 *Support common modularized PET algorithms*. As is indicated in Section 2.2, the PET is a theme of high complexity. Although assumptions about the application PETs in *AdaPrivFlow* will be given in Section 3.1.2, the features of the candidates should be considered in the design. Furthermore, in order to ensure the compatibility and extendability of *AdaPrivFlow*, as well as increase the ease of PET switching, the PETs should be designed a modularized manner, *AdaPrivFlow* must provide sufficient functionality to ensure that common modularized PET algorithms can be correctly applied and switched if required.

R5 *Support common data types in a modern CV*. As is described in Section 2.1, the available data in a CV is characterized by the diversity in the aspects of data type, data dimension and refresh rate. Challenges emerge when data with high dimension and long processing time are processed, such as images. To provide a comprehensive situational privacy protection, *AdaPrivFlow* should be capable of handling the diversity of the characteristics. Qin and Eichelberger [QE16] has proposed the timeliness criterion to assess the quality of algorithm switching. We adopt this criterion in our framework to evaluate *AdaPrivFlow*'s capability of handling the diversity of the vehicle data.

3.1.2 Assumptions

In this section, we will outline *AdaPrivFlow*'s underlying assumption. The assumptions lie in three aspects: the enabling technology, the input vehicle data and privacy policy, as well as the applicable PETs in this framework.

Enabling Technologies

The enabling technologies for *AdaPrivFlow* lie in two aspects: the sufficient computational power from a High-Performance Computing Platform (HPC) onboard, and the supporting hardware-software stack for data collection.

Udo Schifferdecker [Udo20] has given a technical report about the application of HPCs in the automobile industry. One specific application area of HPCs on the automobiles is the data collection. The application of HPCs in the automobile industry enables the tools and languages inherent to the classical IT domains to be incorporated. This insight not only supports the potential of employing the mature techniques from the classic IT domain, but also affirms the feasibility of *AdaPrivFlow* for on-board deployment on a CV.

The other enabling technology is data collecting. There are already existing solutions in this domain. Vector Informatik GmbH has developed its in-house solution [Vecb]. Its application scenario in software-defined Vehicles and its ability of collecting data on demand and publishing to cloud for further analysis corresponds perfectly to the scenario, where our framework *AdaPrivFlow* plays its role. In our framework, it is assumed that a stack of hardware and software of data collection is deployed as the supporting infrastructure.

Vehicle Data and Privacy Policy

AdaPrivFlow is designed to continuously provide querying applications with anonymized vehicle data while adapting to environmental changes. Considering the potential variabilities in the real world, such as sensor malfunctions, as well as fluctuations in the number of third-party applications requiring user data, we established the following assumptions to focus on the development of live adaptation mechanism.

- *The privacy policies are comprehensive.* The privacy policies for evaluating the surroundings are clear, complete and do not contain conflicting settings. A privacy policy is complete if the conditions encountered in the operation are already defined in the policy. The policies are originated either from customized settings or in accordance with legal provisions. The interpretation and creation of these policies are managed by an external component, which is also responsible for the provision of potentially new PETs in the local repository. At this point, we don't evaluate if a privacy policy is appropriate for a given condition. We assume only the comprehensiveness of the definition.
- *The candidate PETs are locally available.* Following the assumption above, we further assume that a PET algorithm is locally available before being supposed to be activated. We do not go further in the situation, how to handle the potential gap if the next PET is not ready to be instantiated.
- *The involved querying applications are constant.* The applications involved in querying remain operational throughout the entire runtime. No application stops or pauses query during the runtime.
- *The queried data sections of a querying application are constant.* The type and the number of the required data section by a querying application remains unchanged during the runtime.

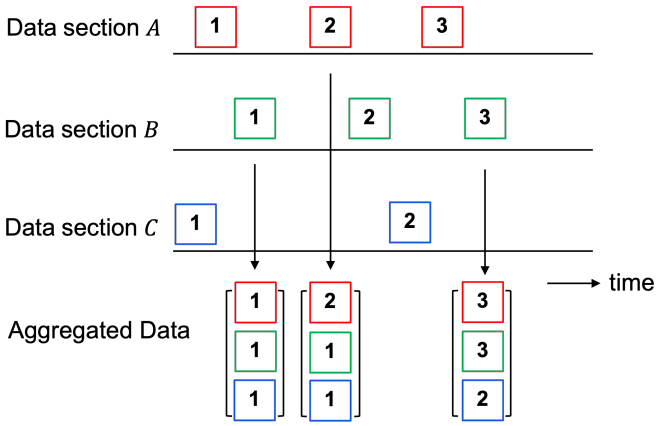


Figure 3.1: An Example of Aggregated Vehicle Data Input

- *The vehicle data are chronologically in order*. The possible out-of-order sensor reading due to fluctuation of transmission inside the vehicle data network are rectified. Conceptually, the central data collector of a CV takes the responsibility. The input of *AdaPrivFlow* follow strictly the chronological order of timestamp embedded in the data.
- *The vehicle data are aggregated*. As is introduced in Section 2.1, the situation is the fusion of (a subset of) the available vehicle data. Consequently, it would be beneficial to have aggregated vehicle data as input. The aggregated is termed as `Vehicle Context`. The aggregated vehicle data reveals the overall state of the ego vehicle, which is always up-to-date. This assumption can be backed up by an intuitive privacy condition, where the situation evaluation is performed with sensor readings from multiple types. For example, a specific PET intended for the speed data section should be activated on the highway, if the speed exceeds 100 km/h. In this case, the evaluator should not only consider the current speed value, but also the geographical location. The aggregation ensures the evaluators for each data section gain access to all necessary information.

We could further assume the approach of aggregation. Figure 3.1 shows an example of the aggregated data with the data records from 3 distinct vehicle data sources. In this visual representation, each square symbolizes an individual data record. Every time one of the data section is updated, a new aggregated input is generated. Hence, the aggregated input always contain the latest data of each data section.

Applicable PET Algorithms

In Section 2.2 the categorization of different PETs is already detailed. Since *AdaPrivFlow* is supposed to be deployed onboard on a CV, it utilizes the *user-side* PETs. Specifically, the primary focus rests on *restructure-based* PETs, such as data perturbation. We further limit the types of restructure-based PETs to be *ego-centric*, which rely exclusively on the sensors onboard, without requiring any data from surroundings or other CVs. The PETs that aggregates the data from other infrastructures or CVs are not considered.

3.2 Conceptual Design

This part illustrates the conceptual development of *AdaPrivFlow*. To start with, in Section 3.2.1 we propose a high-level design of the framework. The essential components in the design are illustrated. In Section 3.2.2, we discuss the characteristics and challenges in the real usage scenarios onboard. The initial design will be optimized so that it can cope with the real-world scenarios. Finally, we integrate the measures and propose the refined data pipeline.

3.2.1 Overview

In this section, we conclude an overview of the data pipeline with the example scenarios in Chapter 1. The main components in the overview and its input and output will be introduced. The data flow and actions in the example scenarios E1 and E2 in Chapter 1 can be generalized as the following process. The CV possesses the knowledge of the privacy policy. The sensors on the CV continuously detect the state of the CV and the environment the CV is located in and generate a flow of vehicle data. The vehicle data in the flow are evaluated against the existing knowledge of privacy policy, Whenever a change in the situation is detected through the situation evaluation, the CV reacts to the change by applying the corresponding PETs. Considering the fact that the situation evaluation and PET enforcement occur simultaneously, the communication between the components in charge of these two computations takes place via signals. Consequently, when a situational change is detected, the component of situation evaluation will notify the component of PET enforcement. The vehicle data is shared after it is processed by the PET. The journey of the vehicle data ends here.

The whole process can be depicted in Figure 3.2. As is shown in the figure, the component at the starting position is the `Input Sources`, which corresponds to the flow of vehicle data as an input. The privacy policy could be usually treated as global knowledge. However, it could also be modeled as a stream. The modeling as a stream has several advantages over global variable. Firstly, as described in Chapter 1, stream processing technology is applied. Modeling the both input as stream is a unified approach. Secondly, although the updates of privacy policies are infrequent, the stream guarantees that even rare modifications are automatically propagated through the system. In contrast, other methods, such as a client polling results, would require periodic checking for updates, introducing a potential delay between the emergence of the update and its processing.

Once ingested into the flow, the vehicle data will be evaluated against the privacy policies in the component `Situation Evaluation`. Should there be any changes in the situation, a corresponding switching decision is generated based on the policy's content. The switching decision is then sent to the `PET Enforcement` in the downstream. It's worth noticing that the specifics of situation

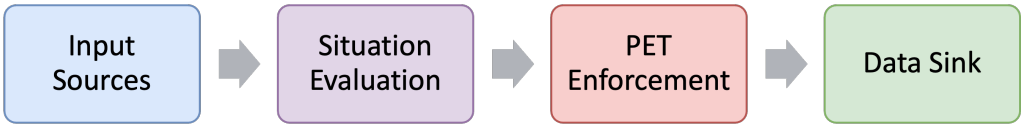


Figure 3.2: The Overview of the Framework's Data Pipeline

evaluation are not part of the central focus of this work. The proposed structure for this segment in the pipeline is primarily intended as an infrastructure that closely mimics a real-world production environment. For a comprehensive theoretical foundation on situation recognition, one can refer to the contribution of Hirmer et al. [HWS+17].

Following the evaluation, the vehicle data is sent to the downstream component `Privacy Enforcement`. Here it undergoes processing by the PET algorithm. The earlier generated switching decision plays the essential role in controlling the action of the `Privacy Enforcement`. When a switching decision is received, the procedure of transition to the new PET is initiated.

Finally, the processed data will be sent to the data sink, where the data is shared with a third-party application. The data sink can be extended to writing to persistent storage for later query or uploading to the cloud. In conclusion, the data that will be eventually accessed by external entities ends its journey in the data sink.

Having outlined the data flow, we now turn our attention to the pipeline's inputs and outputs. As is mentioned earlier, we have modeled the privacy policy and vehicle data as two input streams. Based on the previous work of Li et al. [LHSM23], we proposed an extended definition of the privacy policies, featuring with the provisions on which PETs to use based on conditions. The detailed definition is introduced in Section 4.3.3.

As is described in Section 3.1.2, the input vehicle data are assumed to be aggregated as `Vehicle Context` before being input into the framework. On the contrary, the default outputs are not aggregated. Each data section can be separately queried. This design choice is intentional. A querying application might only be interested in one specific data section. Even if an application requests multiple sections of the data, the alignment of the results might also be unnecessary, since the subsequent processing stages do not require the cross-section correlation. If an application indeed requires this association, it can reconstruct the `Vehicle Context` from the processed data independently.

3.2.2 Architecture

Based on the initial design of the data flow and the strategies to overcome the challenges in on-board application, we present the architecture of *AdaPrivFlow*.

Recall the requirement R2, *AdaPrivFlow* provides distinct privacy enforcement for each querying applications. This requires unique data pipeline for each participating application. Following this idea, the pipeline could be constructed in the system level, as illustrated in Figure 3.3. Each participating application has an independent pipeline, which consists of situation evaluation and PET enforcement. The component `Situation Evaluation` and `PET Enforcement` form the application-specific *privacy zone* together. The application-specific *privacy zone* is the area dedicated to process the data before it's acquired by the environment. Data to be presented to the environment undergoes processing exclusively within the privacy zones. To ensure uniformity of data across these zones, each pipeline receives an identical copy of every input.

The feature of parallel operation is further leveraged in the step of situation evaluation. Zooming in on a single pipeline, the privacy policies designated for each data section could be assessed simultaneously. This is achieved by allocating a distinct situation evaluator for each data section. Figure 3.4 reveals the details. Assuming an application is querying information of 3 data sections,

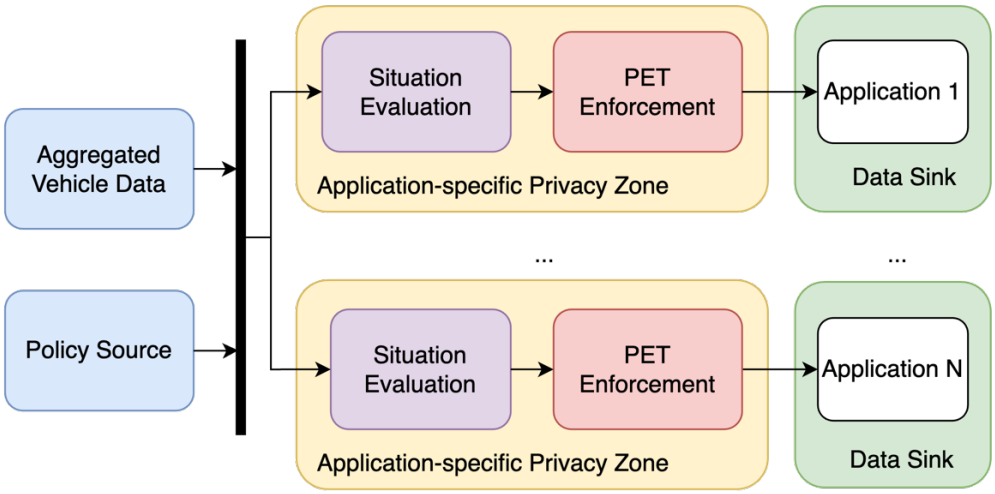


Figure 3.3: Parallel Application-Specific Data Pipeline

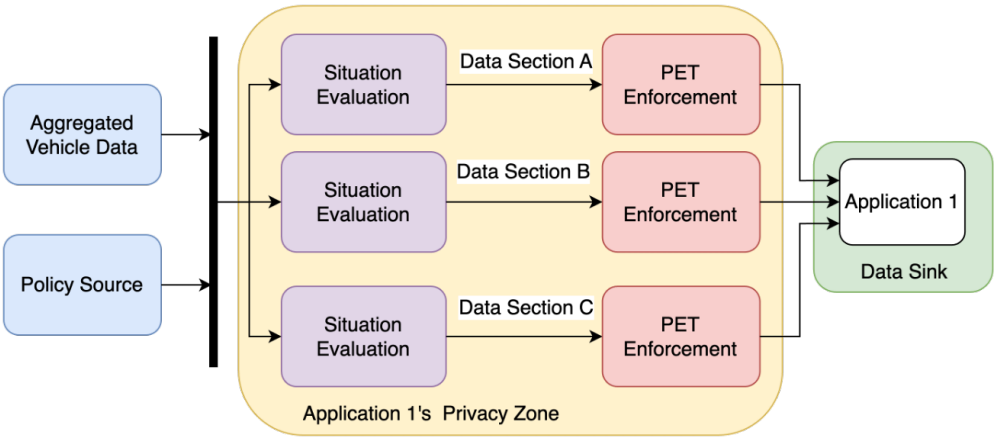


Figure 3.4: Parallel Situation Evaluation and PET Enforcement of One Querying Application

`Data Section A`, `B` and `C`. For each involved data section, a dedicated evaluator is created. Each dedicated evaluator extracts the individual policy of the affiliating data section and evaluates the aggregated vehicle data. The outcome of these evaluations, the switching decision, as well as the assessed aggregated data are relayed to the PET enforcement component relevant to that specific data section. The PET enforcement component also consumes the aggregated vehicle data, allowing the PETs having access to all the necessary information.

Combining Figure 3.3 and Figure 3.4, the architecture is derived and shown in Figure 3.5. In this figure, the layered boxes indicate the multiple instances of each component. The Figure 3.5 depicts a typical pipeline, where there are two querying applications. The two stapled yellow boxes are the individual *privacy zones* for each application. One of the application seeks the information from 3 data sections. The current aggregated vehicle data are evaluated by the dedicated evaluator against

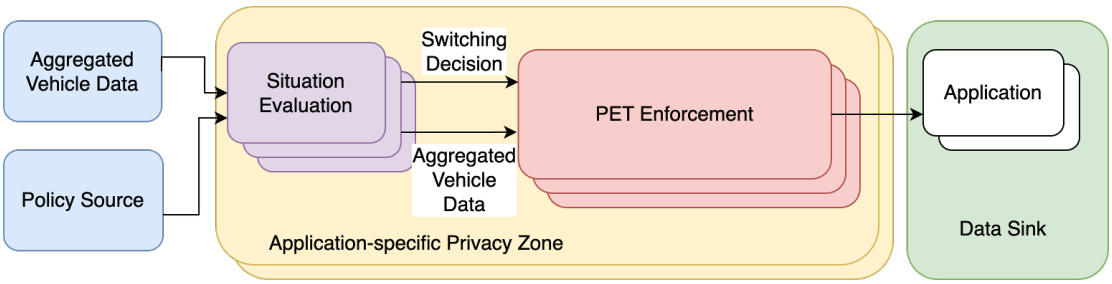


Figure 3.5: Architecture of the Framework

the policy specifically for each data section. A particular data section requires distributed PET enforcement, hence this segment of pipeline is equipped with parallel instances of PET enforcement operators.

3.3 PET Adaptation: Core Concept

In this section, we detail the mechanism of PET adaptation. At first, we explore different approaches of PET adaptation, before a design decision is fixed. Following this, we will break down the adaptation procedure into several steps for a more in-depth understanding.

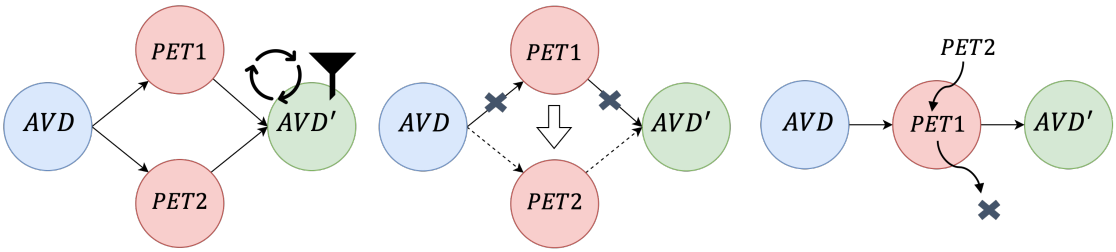
3.3.1 Approaches

Figure 3.6 illustrates three potential strategies to dynamically adapt the PET algorithm. The graph representation is used to illustrate the structure of the pipeline. In the topology, circles represent nodes, which can either be a source, an operator, or a sink. The process of applying a PET algorithm can be viewed as an operator in the topology. The topologies in Figure 3.6 are the minimal pipelines of a job that processed the aggregated vehicle data (AVD) with PET. For consistency, the color scheme from in Figure 3.3 to Figure 3.5 is retained. The nodes in light blue represent the source of aggregated vehicle data. The nodes in red are responsible for PET enforcement. The green nodes depict the sink that receives the processed vehicle data in the aggregated form (AVD'). Three aspects should be considered when making the design choice:

- *Capability of handling dynamic policies.* As is pointed out in the scenarios in Chapter 1, a CV might obtain fresh mandatory privacy policies from the environment. These can introduce locally unavailable PETs. Although based on our assumption, other components take responsibility of supplying these PETs, and we do not delve into the detail in this part, our framework's pipeline must efficiently utilize the appropriate PET once provisioned.
- *Computational Resource Consumption.* In Section 3.1.2, we have assumed the application of HPCs in the deployment of *AdaPrivFlow*. However, even with a HPC on board, the computational resources are finite. Excessive consumption could influence the responsiveness and the energy efficiency of the onboard computing system, restricting its potential of being extended with more functionalities and resulting in higher demand for computation hardware.

- *Switching Complexity.* If a switching procedure has lower complexity, it will have a positive effect on the down-time and the consumption of computational resource. Furthermore, the cost of handling data and state should be lower and less error-prone.

Based on these aspects, the three potential approaches in Figure 3.6 are observed.



(a) Variant 1: Output Filtering **(b)** Variant 2: Dynamic Pipeline **(c)** Variant 3: Dynamic PET Loading

Figure 3.6: Concepts of PET adaptation

Variant 1: Output Filtering

This approach is depicted in Figure 3.6a. In this case, all the candidate PET algorithms participate in the computation. The switching decision imposes its influence in the data sink. The data sink processes the switching decision and filters out the processed results from irrelevant PET candidates. This approach works with a *static* pipeline. Once started, the participating PETs is fixed with regard to the policy definition. This approach clearly struggles with dynamic policies during the runtime. Nevertheless, the switching complexity is rather low. The sink needs merely to adapt its filter criteria. Despite the low complexity, the approach demands higher computational resource, since all the candidate PET algorithms participate in the computation, regardless of whether they are currently applicable. In this way, lots of computational resources are wasted on useless results. The situation would become even worse, if a data section needs to be processed by multiple parallel instances of PET enforcement operator.

Variant 2: Dynamic Pipeline

This approach is depicted in Figure 3.6b. It works based on the principle of a *dynamic* pipeline. In this approach, every candidate PET is built in one dedicated operator. Upon receiving a switching decision, the operator of the current PET is substituted by the one of the next PET. As is shown in Figure 3.6b, the connection with the operator of PET2 will replace the one of PET1. The topology will be changed in one switching event. Clearly, during the runtime, only the operators with the applicable PET algorithms participate in the computation. Therefore, there is no extra consumption of computational resources. This approach is intrinsically capable of handling dynamic policies. However, the switching procedure is the most complicated among the three approaches. Upon receiving the switching instruction, the entire stream processing job should be paused, as of the current development of stream processing technology. The down-time allows the pipeline to be

rebuilt with the operator mounted with the next PET algorithm. The job is then resumed after the pipeline is rebuilt. Since this switching procedure involves the restart of the stream processing job, much attention should be paid to the management of the data and the internal states of the system.

It's further worth noticing that a *dynamic* pipeline is the solution for the shortcoming of Variant 1. With a dynamic pipeline, the topology could be rebuilt by integrating the next PET into it.

Variant 3: Dynamic PET Loading

This approach is illustrated in Figure 3.6c. In contrast to Variant 2, the PETs share one `PET Enforcement` operator. This uniqueness demands a common interface for the compatible PETs. They should be designed in a modularized manner The `PET Enforcement` operator controls the lifetime of the PET inside. As is shown in Figure 3.6c, the operator loads the PET2 upon receiving a switching decision. The algorithm PET1 is discarded. During the switching procedure, the streaming processing job remains active. Clearly, this approach operates in a *static* pipeline. Nevertheless, it's capable of dealing with dynamic policies through loading different PETs in the `PET Enforcement` operator. In this approach, only the current applicable PET algorithms consume the computation resources. Therefore, there is no extra cost for computation. The complexity of switching lies in the loading mechanism. Although it is not as trivial as adapting the filter criterion in Variant 1, it is still much simpler than manipulating the whole pipeline like in Variant 2. Given its balance between adaptability and complexity, it is chosen to be the foundational approach for the further development.

In the following, we will take a closer look at the switching procedure of Variant 3. The switching procedure can be decomposed into two main steps: (a) Upon receiving the switching decision, load the next PET algorithm, and (b) Set the next PET algorithm as active. The details of the step (a) will be introduced in Section 3.3.3 and step (b) in Section 3.3.3.

3.3.2 Provision of PETs

Based on the analysis in Section 3.3.1, the approach of realizing the adaptation is determined as Variant 3 in Figure 3.6. The first issue that needs to be addressed is the approach for algorithm instantiation.

Figure 3.7 illustrates the components and steps for provision of PETs inside the `PET Enforcement` operator. As is depicted in Figure 3.2, a `PET Enforcement` operator receives the switching decision and the aggregated vehicle data. A `PET Enforcement` operator has one working PET algorithm, which ingest the input of vehicle data and apply privacy protection. The ingestion of a switching decision activates the switching procedure. The most important message contained in a switching decision is the annotation of the PET to be applied. The annotation is written in a machine-readable form and includes metadata about this PET, including its name, its property of being stateful or stateless, as well as the input parameters for initialization. As the first step, the component `Annotation Processor` parses the annotation. An example of such annotation is provided in Listing 4.1. In the second step, the parsed information is relayed to the `Algorithm Provider`, which looks up for the corresponding algorithm in the local algorithm repository and initialize it with the parameters. The third step of switching doesn't necessarily take place right after the completion of initialization. The

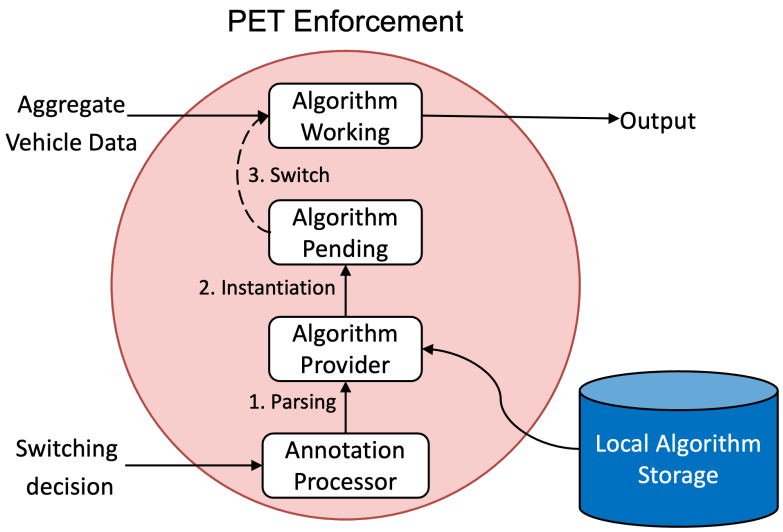


Figure 3.7: The Lifetime of a PET Algorithm in the PET Enforcement Operator

newly instantiated PET is set to be pending temporarily. Once it is feasible, the newly instantiated PET algorithm will be set to active, substituting the last working PET. The condition of feasibility is explained in Section 3.3.3.

3.3.3 Feasible Condition of PET Switching

In this section, we will take a closer look at the third step in Figure 3.7, the switching procedure. The first two steps of parsing and instantiation are already covered in Section 3.3.2. Recall the requirement R3 proposed in Section 3.1.1, in order to prevent potential privacy breach, the PET should be accurately applied on the vehicle data, according to definition in the established policies. In order to fulfill this requirement, the switching action must occur under the premise that a certain condition is met. This condition is termed as *feasible condition*.

Before we derive the feasible condition for on-time switching, the I/O state of a `PET enforcement` operator should be understood. As is described in Figure 3.5, the operator of PET enforcement has two input channels. In one channel the data records to be enforced with PETs streams into the operator. The other channel receives the switching decision from the upstream `Situation Evaluation` component. The elements in the two channels flow in parallel into the `PET enforcement` operator. Figure 3.8 gives a snapshot of input and output channel PETs, reflecting the general I/O situation.

In Figure 3.8, a PET algorithm $\mathsf{PET}(\cdot)$ is currently active inside the operator. The boxes marked with a_i are the aggregated vehicle data in a stream. The ascending subscripts indicate the chronological order of the data generation. Recall the assumption in Section 3.1.1, the ingested data are chronologically in order. Therefore, their ingestion order corresponds to their generation order. In other words, an earlier generated aggregated vehicle data will also be ingested earlier. The shadowed box with the label a_t marks the first data record that satisfies a different condition defined in the policy, meaning that a_t and the records afterwards comply to a new PET, while the data

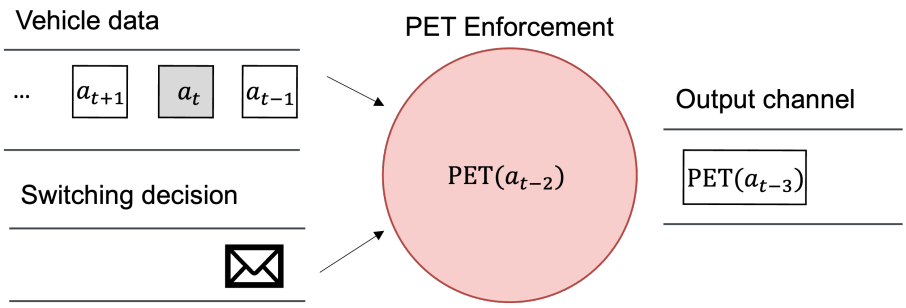
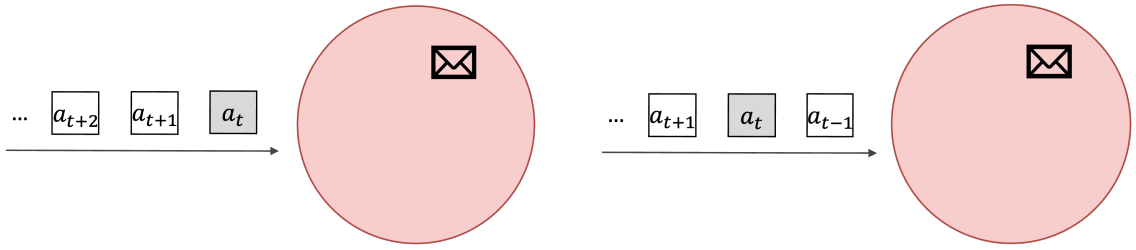


Figure 3.8: I/O Snapshot of a PET Enforcement Operator. a_i represents the aggregated vehicle data. PET$(\cdot)$ is the working PET inside the operator.

records before this trigger should be processed with the current one, i.e., PET$(\cdot)$. Therefore, a_t triggers the switching procedure. In the following, it will be called as the *trigger*. The envelope icon represents the switching decision evaluated based on a_t.

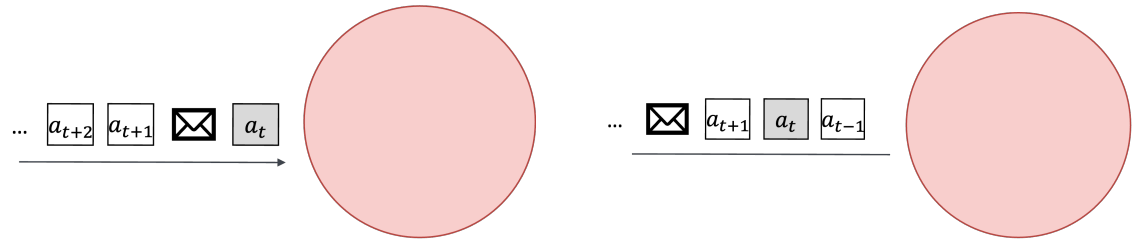
In the depicted I/O state in Figure 3.8, if the next PET algorithm is to enforce accurately on a_t and the data records afterwards, the operator should perform the operations in the following order: (a) consumes a_{t-1} and emits PET(a_{t-1}), (b) consumes the switching decision and prepare the next PET following the steps 1 and 2 described in Figure 3.7, (c) right after the instantiation, activates the next PET as the 3rd step in Figure 3.7, and (d) consumes a_t and the data afterwards, processing them with the next PET. However, the nature of distributed computing of a SPS should be kept in mind. In the default deployment mode, the data transmission undergoes the network stack to reach the distributed computing nodes. This results in a non-deterministic arrival of the input elements and even dropping of messages. *AdaPrivFlow* is deployed in standalone mode, the data is transmitted in memory and therefore no dropping of the messages. Nevertheless, each operator accesses the memory also in a non-deterministic manner. Although an order of consumption could be defined deliberately on the side of the operators, this approach would bring negative influence to the throughput of the system, since a gap between arrival and consumption is very likely to appear. Therefore, without loss of generality, the operators in *AdaPrivFlow* is supposed to consume the input elements in a non-deterministic order. The proposed requirement R3 originates from this phenomenon. In the scenario in Figure 3.8, if no measure is taken, the requirement R3 cannot be guaranteed, since the switching decision could be processed before or after an arbitrary element in the input channel of aggregated vehicle data.

The approach to ensure accurate PET enforcement is to introduce the *trigger marker* and the *buffer*. The *trigger marker* makes the *trigger* identifiable. The *buffer* is a functional component inside the PET enforcement operator and follows the First-In-First-Out (FIFO) principle. The idea is to make PET enforcement operator aware of the *trigger*, even with the absence of the associated switching decision. Therefore, the *trigger* should be identifiable. The trigger marker alone is not adequate to solve the problem. Even if the *trigger* is identified, it should be held until the affiliating PET is successfully prepared. This can be achieved by utilizing a buffer. In Figure 3.9, the collaboration between the marker and the buffer are discussed in 4 different cases, based on the processing order of the elements in each channel. The cases are distinguished based on two facts: (1) if the `PET Enforcement` operator has already processed a switching decision, and (2) if the next aggregated vehicle data is a *trigger*.



(a) The switching decision: processed
Next AVD: the trigger

(b) The switching decision: processed
Next AVD: non-trigger



(c) The switching decision: not processed
Next AVD: the trigger

(d) The switching decision: not processed
Next AVD: non-trigger

Figure 3.9: Different cases of a PET enforcement operator handling aggregated vehicle data (AVD). In each sub-figure, the elements are ingested chronologically from the right to the left.

1. *Handling trigger **with** processed switching decision.* Figure 3.9a depicts the scenario, where the switching decision is already processed by the operator. The *trigger* is next element to be processed. This scenario is the simplest situation. Without any additional steps, the next PET can be set to active. The *trigger* and afterwards are enforced with the correct PET.

2. *Handling **non**-trigger **with** processed switching decision.* In Figure 3.9b, the next data record to be processed is non-trigger, while the switching decision is already processed by the operator. This case may occur if the current PET is a long-running algorithm, resulting in queuing of data records waiting to be processed. In this case, the newly instantiated PET will be held temporarily in pending state. Until the trigger appears, the data records should be processed with the current PET. When the trigger is ingested by the operator, the pending PET will be set to active state.

3. *Handling trigger **without** processed switching decision.* As is shown in Figure 3.9c, by the time the *trigger* is about to be ingested, the operator is unaware of the upcoming switching event, because the switching decision is delayed. At this point, the *trigger* should be buffered until the switching decision is processed, so that it is prevented from being processed with the current PET.

4. *Handling **non**-trigger **without** processed switching decision.* The scenario in Figure 3.9d is the most common situation during the operation. On the one hand, the next data record to be processed is non-trigger. On the other hand, The operator is unaware of any switching decision. There are two possibilities. Firstly, during normal execution, there is no upcoming switching event and the records are enforced with current PET. In this case, nothing special happens. a_{t-1} faces exactly this situation. Secondly, the consumption of the switching

decision is delayed, while the trigger is already processed. This situation applies to any records being processed between the trigger and the switching decision, such as a_{t+1} in Figure 3.9d. In this situation, the scenario in Figure 3.9c has already taken place, since the trigger a_t has already been consumed and held in the buffer. The data records afterwards should be buffered. The buffer will not be released until the affiliating switching decision is successfully processed.

After analyzing the possible situation due to non-deterministic arrival of elements, the feasible condition of PET switching can be obtained: If the switching decision is processed and the next element to be ingested is a trigger, the condition is met and the next PET can be set to active state.

The four described behavior of a `PET Enforcement` operator can be expressed in the form of pseudocode in Algorithm 3.1. The handling procedure is called `Handle`() and takes in one aggregated vehicle data as input argument, which is denoted as a. This procedure is invoked every time an aggregate vehicle data is ingested.

As is shown in Algorithm 3.1, the state of buffering is defined as a global variable in line 1. It could be altered within the procedure. The procedure judges the situation in two steps. First, it judges if the switching decision is processed. The outcome falls either in the first row or the second row in Figure 3.9. Then, it checks if the next element of aggregated vehicle data is a *trigger*. Line4 to line 7 corresponds to the situation depicted in Figure 3.9a. With existing switching decision and *trigger*, the feasible condition is met. The actually switching takes place by calling the method `Switch` in

Algorithm 3.1 PET enforcement operator handles vehicle data records

```
1:  bufferMode ← false
2:  procedure HANDLE(a)
3:      if Switching decision is processed then
4:          if a is trigger then                                  // Situation in Figure 3.9a
5:              SWITCHPET()
6:              bufferMode ← false
7:              emit ENFORCEPET(PETWorking, a)                    // PETWorking is switched.
8:          else                                                  // Situation in Figure 3.9b
9:              emit ENFORCEPET(PETWorking, a)                    // PETWorking is not switched.
10:         end if
11:     else
12:         if a is trigger then                                  // Situation in Figure 3.9c
13:             bufferMode ← true
14:             Put a into the buffer
15:         else                                                  // Situation in Figure 3.9d
16:             if bufferMode is true then                        // Refer to a_{t+1} in Figure 3.9d
17:                 Put a into the buffer
18:             else
19:                 emit ENFORCEPET(PETWorking,a)                 // Refer to a_{t-1} in Figure 3.9d
20:             end if
21:         end if
22:     end if
23: end procedure
```

Algorithm 3.2 Switch to next PET

Require: PETPending non-null
1: **procedure** SWITCHPET()
2: PETWorking ← PETPending
3: **while** buffer ≠ ∅ **do**
4: pop the first record e in the buffer
5: emit ENFORCEPET(PETWorking, e)
6: **end while**
7: **end procedure**

line 5. The concrete steps in this method are given in form of pseudocode in Algorithm 3.2. The method `EnforcePET` invokes the intrinsic method in the PET algorithm of processing the aggregate vehicle data. The result is then emitted to the downstream. Details about the intrinsic methods will be illustrated in Section 4.3.1.

The switching procedure is given in Algorithm 3.2. Recall Figure 3.7, the newly instantiated PET algorithm is set to pending and waiting for the feasible condition for the switching to be met. In line 2, the pending new PET replaces the current active PET. Considering the cases in Figure 3.9b and Figure 3.9d, where the aggregated vehicle data is buffered because of the delay of switching decision, the buffer should be released after the actual switching of PET is completed. The release of buffer can fulfill the requirement R3, because of two reasons: (1) the buffer follows the FIFO principle, and (2) the buffering mode is activated only after the *trigger* is ingested (ref. Figure 3.9c and line 12 to line 14 in Algorithm 3.1).

3.4 PET Adaptation: Refinement

In Section 3.3, we have discussed the potential approaches to realize the adaptation of PET in the data pipeline. The design choice is set to be dynamic loading of PETs in a static data pipeline, as is shown in Figure 3.6c. Based on the principle of this variant, we proposed provision procedure of the PET algorithm, as illustrated in Section 3.3.2. Aimed at fulfilling the requirement R3, we elaborated the switching mechanism. In this section, the proposed approach of PET provision and switching will be checked against the rest of the requirements. Requirement R1 and R2 are not related to this aspect and thus excluded from the examination.

3.4.1 Limitation and Optimization Potentials of Core Concept

Through the examination of requirement R4 and R5, following limitations of the proposed approaches can be concluded.

Firstly, the proposed method of PET provision didn't cover the reliance of history records in *stateful* PETs. As is pointed out in Section 2.2, *stateful* PETs process the input with established information. If a stateful PET is initialized in a cold manner, meaning without the history records, it takes indefinite time to be able to provide privacy protection, since it needs to wait for sufficient incoming data to establish the necessary information. This process is termed as *warm-up* in this work. The warm-up process can be accelerated if a stateful PET can be provided with the necessary data upon

instantiation. To achieve this, *AdaPrivFlow* should have the access to the repository of history data in the CV's software stack. However, the management of history data, including the frequency of backup and the determination of Time to Live (TTL) lies in the responsibility of the dedicated service inside the CV's software stack. Besides, it is mentioned in Section 2.2 that the necessary history data may originate from different CVs. Although in the current design of *AdaPrivFlow*, only ego-centric PETs lie in the focus. Nevertheless, considering the potential of expanding the scope of compatible PETs to the non-ego-centric class, a repository of history data also helps store the data from other entities acquired through information exchange. The integration of such a repository could be extended to the integration of a general purpose *persistent storage*. Not only the aggregated data can be stored there, but also the established policies. Furthermore, the local algorithm repository mentioned in Section 3.3.2 could be placed inside this persistent storage. The `PET Enforcement` component in Figure 3.7 accesses the database to fetch the PET algorithm and to prepare the stateful PET, if necessary. In summary, the persistent storage keeps the following contents:

- *Last Privacy Policy*. The last applicable privacy policy is stored in the database, so that the framework can start with a defined policy in the next run.
- *PET Algorithms*. The PET algorithms defined in the privacy policy are stored in a local repository, so that they can be fetched by a dedicated component for switching inside the `PET Enforcement` component.
- *History Vehicle Data Records*. In addition to the proposed aim in [LHSM23] to meet the regulation of law enforcement, the stored data also accelerate the state building procedure for stateful PET algorithms.

Secondly, the characteristics of the available vehicle data in a CV is not considered in the proposal of switching mechanism. As is mentioned in Section 2.1, the data subjected to privacy preservation vary in dimensions and refresh rates. High dimensional data, such as the image data, inherently contain larger volumes of information. Besides, they are refreshed at higher frequency [Pim17]. These types of data are also frequently processed with complex algorithms like neural networks. These facts contribute to the increased latency of processing, and even worse, an unbounded queue of such data types.

Real-world use cases of querying such high dimensional data is prevalent. Dettinger et al. [DWW+23] examined how cooperative efforts among sensing participants can be used to form a comprehensive environmental perception. In their work, the vehicle data is preprocessed onboard and shared via mobile network data exchanges. The actual computation happens outside a CV. Techniques like semantic segmentation of camera data and sensor fusion between cameras and LiDAR are employed in these scenarios. This approach demands low processing latency to realize prompt detection of the dynamic objects, such as vehicles and pedestrians. Another typical scenario of querying complex data is the widespread usage of dashcams. While dashcam video sharing has grown in popularity for accident investigation and entertainment purposes, there is increasing concerns about expanding urban monitoring [PKML16]. There are already existing work of utilizing the vehicular network and onboard camera, such as for the purpose of tracking and monitoring suspicious vehicle [CTS14]. These application scenarios require not only low processing latency, but also the correct order of the frames.

Our approach to mitigate the processing latency is to deploy multiple parallel instances with the same PET, which run in parallel. These instances concurrently access the input channel and emit the processed result in the shared output channel. Each instance a singular incoming data record and enforce the PET algorithm on this data. While individual data records are processed by unique instances of PET enforcement operators, the records are collectively processed in a distributed and parallel manner. Later in this thesis, this method is referred to as *distributed PET enforcement*. To differentiate, the counterpart of this method, where the PET enforcement is performed by a single operator, is termed as *centralized PET enforcement*.

Until now, the categories regarding the state of PET and the number of operators are established. However, they describe the PET enforcement schema on two distinct dimensions. The relation between these two dimensions is given as the following:

- *Stateless Centralized PET Enforcement*. These are PETs dealing with data requiring low computation resource without needing prior information to produce the result. Such PETs do not need to be optimized.
- *Stateful Centralized PET Enforcement*. Such PETs utilize established information to process the vehicle data. The strategies of accelerating the state-building phase is to allow database access. In this way, such stateful PETs do not need to wait for future incoming data to reach their internal states.
- *Stateless Distributed PET Enforcement*. The PETs in this category handle high-dimensional data or is long-running. They don't rely on established information.
- *Stateful Distributed PET Enforcement*. The establishment of a state should be performed in one unique instance of operator. If it should be performed across multiple instances, cross-instance information exchange is necessary, leading to potential issues such as instance synchronization and increased communication latency. Essentially, a stateful PET which need distributed enforcement actually build states for several aspects simultaneously. This can be reduced to several parallel-running centralized enforcement of multiple stateful PETs.

Apparently, *Stateless Centralized PET Enforcement* doesn't suffer from any insufficiency mentioned above. It can be applied with the core concept of PET switching in Section 3.3 without problem. In Section 3.4.2, the refinement for *Stateful Centralized PET Enforcement* is introduced. In Section 3.4.3, the refinement for *Stateless Distributed PET Enforcement* is illustrated. As is already mentioned, *Stateful Distributed PET Enforcement* could be decomposed into multiple parallel-running jobs of *Stateful Centralized PET Enforcement*. Therefore, the discussion of this type is not necessary.

3.4.2 Centralized Stateful PET Enforcement

Recall the limitation mentioned in Section 3.4.1 about the stateful PETs, the limitation lies in the fact that the proposed instantiation process didn't consider their reliance on the existing data. The feasible condition of PET switching, which is introduced in Section 3.3.3, can be applied to this category without problem. In the following, we introduce the refinement measures regarding this limitation.

As is already proposed in Section 3.4.1, a persistent storage could be supplied to provide the stateful PETs with existing data while being instantiated. The interaction with the persistent storage during the instantiation is shown in Figure 3.10. As is shown in the figure, the step of storage access takes place after the next PET algorithm is switched to be active. Keeping the requirement R4 in mind, a `Storage Agent` is employed. This component takes the responsibility of fetching the necessary data. Through this design, the details of the configuration and access are hidden from the PETs. The `Storage Agent` operates based on the information provided by the `PET Enforcement` operator. To achieve this, the `PET Enforcement` operator relays the relevant information in the PET annotation to the `Storage Agent`. Details about this point will be illustrated in Section 4.3.2, where the documentation of the PET's dependency on existing data will be illustrated.

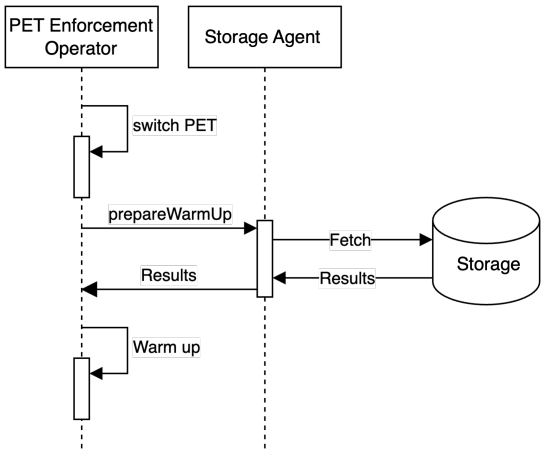


Figure 3.10: Storage Access during Instantiation of Stateful PETs

With consideration of preparation of stateful PETs, the switching procedure in Algorithm 3.2 can be modified to Algorithm 3.3. Line 3 to line 6 add the optional step of storage access for stateful PETs. Line 9 to line 12 takes the situation into account, where the fetched data is still insufficient for the stateful PET to establish its internal state. In this case, the PET utilizes the current incoming data to warm itself up. Algorithm 3.1 is extended to Algorithm 3.4. In line 3, the state of the PET will be evaluated. In this way, the PET will utilize the incoming state afterwards to warm itself up, until the information is sufficient. After the warm-up phase is complete, the PET continues to process the incoming aggregated vehicle data *a* by invoking the method `Handle` in line 7.

3.4.3 Switching in Distributed PET Enforcement

In Section 3.4.1, we proposed *distributed PET enforcement*, regarding the challenges brought by high dimensional data, where low processing latency and ordered data are demanded. However, the feasible condition of switching is not applicable for this case, since the introduction of multiple instances of `PET Enforcement` operators alters the way of data ingestion. In the *centralized* enforcement schema, all the elements in a stream are processed definitely by one single operator. In the *distributed* enforcement schema, each element is processed by a different instance of the

Algorithm 3.3 Switch PET considering Reliance on Other Information

```
Require: PETPending non-null
 1: procedure SwitchPET()
 2:     PETWorking ← PETPending
 3:     if PETWorking is stateful then
 4:         Delegate data retrieval to the database agent
 5:         Warm up PETWorking
 6:     end if
 7:     while buffer ≠ ∅ do
 8:         pop the first record e in the buffer
 9:         if PETWorking is not ready then
10:             Warm up PETWorking
11:             return
12:         end if
13:         emit EnforcePET(PETWorking, e)
14:     end while
15: end procedure
```

Algorithm 3.4 PET enforcement operator handles vehicle data records considering stateful PETs

```
1: procedure Handle_State(a)
2:     bufferMode ← false
3:     if PETWorking is not ready then
4:         Warm up PETWorking
5:         return
6:     end if
7:     Handle(a)
8: end procedure
```

operator. Distributed PET enforcement brings two issues as consequence: *unsynchronized switching* and *out-of-order output*. In the following, we explore the causes of these phenomena and propose countermeasures.

Out-of-Order Output

The reason for this phenomenon lies in two aspects. Firstly, the execution order is non-deterministic. Each parallel instance of the PET enforcement operator could be operated on different CPU cores. The execution order is non-deterministic and dependent on the scheduling of the CPU tasks. Moreover, the processing time of each single data record can vary. Without appropriate countermeasures, the downstream operator will receive a stream of data processed out of order.

The solution of keeping the order after processing is to define a sequence of dispatching to each parallel instance. The sequence is defined through a mapping of a set of special attribute (called *keys*) to the index of the parallel instances. The total number of the keys equals the total number of parallel instances. As long as the keys are iterated in a deterministic order, the dispatching sequence is deterministic. A complete iteration of the keys is termed as a *dispatching cycle*. The

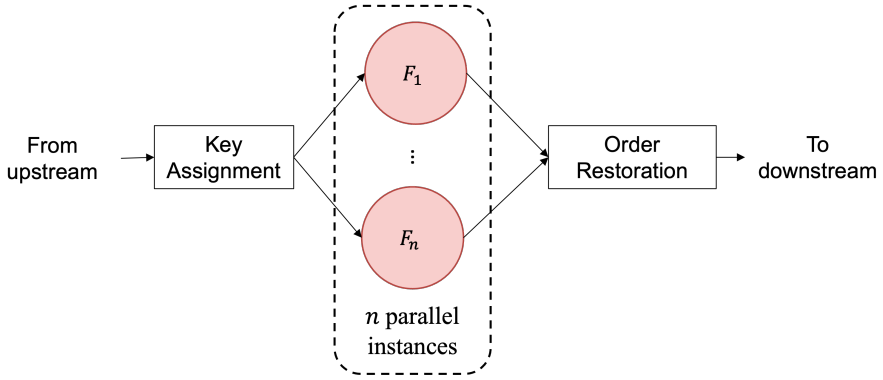
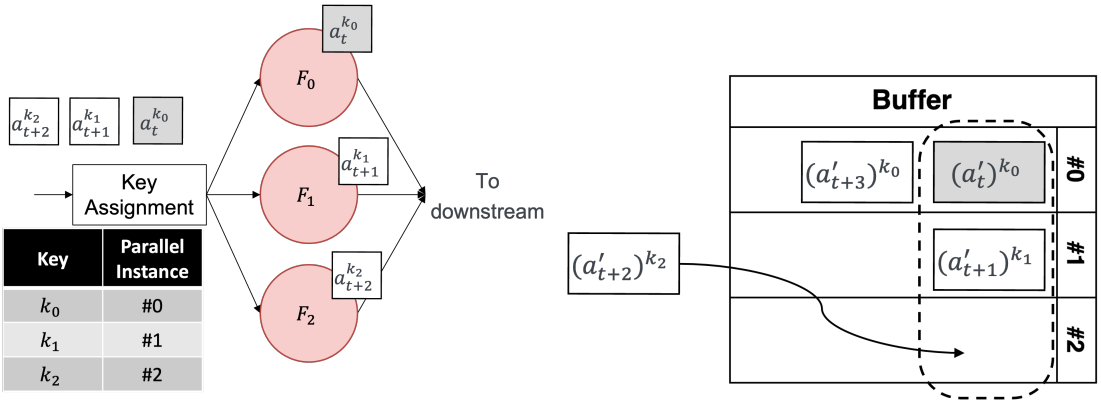


Figure 3.11: Concept of Distributed PET Enforcement with Data Order Keeping



(a) Deterministic Dispatching to Parallel Instances by Keys

(b) Order Restore based on Keys

Figure 3.12: Example of Distributed PET Enforcement with Data Order Restore

order restoration of the processed data can be completed by just following the same key sequence, which requires that the processed data also carry the keys. The processed data should be buffered temporally before being emitted to the downstream. The emission takes place only if all the processed data of the same dispatching cycle are buffered.

This pipeline for this approach is proposed in Figure 3.11. As depicted in this figure, before the data records are dispatched to the parallel instances F_i, each of them is tagged with a key by the `Key Assigner`. The keys are tagged to the data records without modifying their contents. An example for the illustration is provided in Figure 3.12a. As the data a_i flow through the `Key Assigner`, they are tagged with the keys $k \in \{k_0, k_1, k_2\}$. According to the mapping relation in the example, a_t will be dispatched to F_0, and so on. After the data are processed, they flow through the `Order Restoration` operator in Figure 3.11. The details of order restoration is illustrated in Figure 3.12b. The boxes represent the same data in Figure 3.12a, except being processed, as indicated with the prime symbol ($'$). For each parallel instance, a FIFO queue is maintained. The processed data are pressed into the corresponding queue, according to key they carry. For example, $a_t^{k_0}$ is processed by F_0, as is indicated by the key k_0. Hence, $(a_t^{'})^{k_0}$ is pressed into the FIFO queue of F_0. As is shown in Figure 3.12a, the *dispatching cycle* is #0 $\rightarrow$ #1 $\rightarrow$ #2. Figure 3.12b depicted a scenario where the arrival of the processed data are out of order. Here $(a_{t+3}^{'})^{k_0}$ arrives earlier than $(a_{t+2}^{'})^{k_2}$. After

$(a'_{t+2})^{k_2}$ is added into its corresponding queue (#2), the top positions of the queue are fully occupied. The data in this dispatching cycle are ready to be emitted. They are popped out of the FIFO queue in the ascending order. In this way, the chronological order of the output records are preserved.

Unsynchronized Switching

Unsynchronized switching refers to the scenario, where not all the parallel instances have successfully launched the new PET before each instance starts processing the data complying to the new PET. This violates the requirement R3. This phenomenon is depicted in Figure 3.13. In this figure, each parallel instance of the `PET Enforcement` operator F_i is able to receive the same switching decision. They share the same input channel of the aggregated vehicle data. In the depicted scenario, a_t, a_{t+1} and a_{t+2} are processed by F_0, F_2 and F_3, respectively. This assignment of elements to instances is merely an example among other possibilities, as long as one element is processed by one distinct instance.

Under the circumstance of distributed PET enforcement, the feasible condition of switching from Section 3.3.2 can no longer ensure the fulfillment of requirement R3. Recall the feasible condition, the PET enforcement operator identifies the *trigger* and buffers the aggregated vehicle data afterwards until the next PET is successfully switched. However, in the case of distributed PET enforcement, the *trigger* is consumed by only one instance of the operators. Take the scenario in Figure 3.13 as an example. F_0 is able to detect the *trigger* a_t and follow the principle of feasible condition. On the contrary, F_1 and F_2 are completely unaware of the trigger. In case of delayed switching decision, F_1 and F_2 wouldn't take any action. if the feasible condition were applied. Therefore, a_{t+1} and a_{t+2} would be processed with the wrong PET.

To solve this problem, two measurements are taken. The solution is presented in Figure 3.14. Firstly, a dedicated operator, the `Controlled Buffer`, is added in the upstream of the parallel instances for PET enforcement. The trigger and the data afterwards will be intercepted upon the trigger is identified. Secondly, to synchronize the states among the parallel instances, a feedback mechanism is introduced. The parallel instances emit a `READY` signal back to the upstream, reporting that the switching decision has been successfully processed and the next PET is ready. Once the feedback

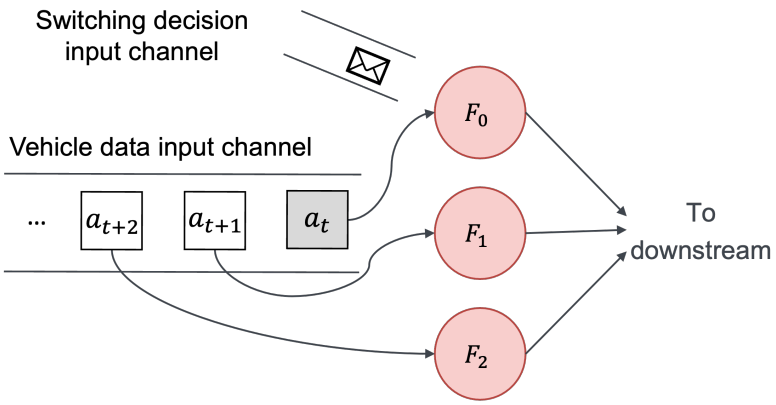


Figure 3.13: Input Consumption of Multiple Parallel Instances. F_i are parallel instances of a PET enforcement operator. a_i are aggregated vehicle data.

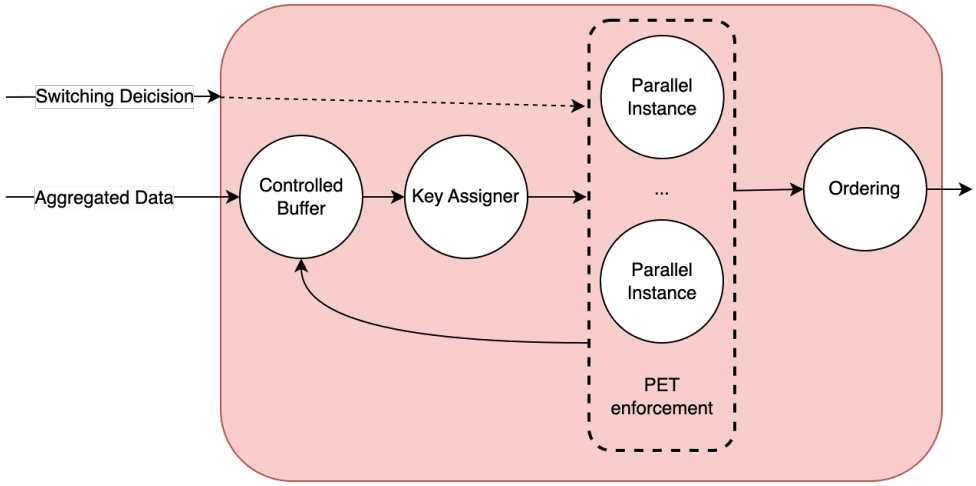


Figure 3.14: Accurate PET Enforcement Pipeline for Multiple Parallel Instances

from all the parallel instances are collected, the buffered triggering data and afterwards are safe to be released to the downstream. The feedback signal is collected and processed by a dedicated operator called `Feedback Processor`. The buffer releases the data under the instruction of the `Feedback Processor`, which is the reason for the name `Controlled Buffer`. Then the data go through the procedure of order keeping mentioned in the previous section. In this way, the processing of data sections with high processing expenditure can be accelerated while fulfilling the requirement of R3. Until now, the conceptualization of *AdaPrivFlow* is completed. We have established an architecture with 4 essential components to perform the task of PET adaptation based on situational evaluation. Considering the needs and challenges in real-word application, on the one hand, we enriched the pipeline with the access to database. On the other hand, we categorized 3 realistic PET enforcement patterns. Regarding each pattern, we proposed the refined data pipeline with extra measures to ensure the framework works efficiently, and enforce the PET accurately according to the established input policies on the theory level.

3.5 Summary

Based on the overview in Figure 3.2, we conceptualized the architecture in Figure 3.5, which contains 4 components: `Input Source`, `Situation Evaluation`, `PET Enforcement` and `Data Sink`. As illustrated in Section 3.4, the characteristics of stateful PETs and vehicle data with high dimension are analyzed. Based on the challenges they bring by algorithm switching and PET enforcement, we not only integrated the persistent story into the pipeline, but also proposed distributed PETs enforcement to handle the data with higher dimension. Combining these measure with the architecture in Figure 3.5, we derive refined architecture in Figure 3.15. Compared with the original architecture, the refined architecture highlights the integration of the persistent storage and its interaction with the components in the pipeline. The `PET Enforcement` components highlights the proposed distributed PET enforcement method by showing the `Parallel Instances`. With the refined architecture

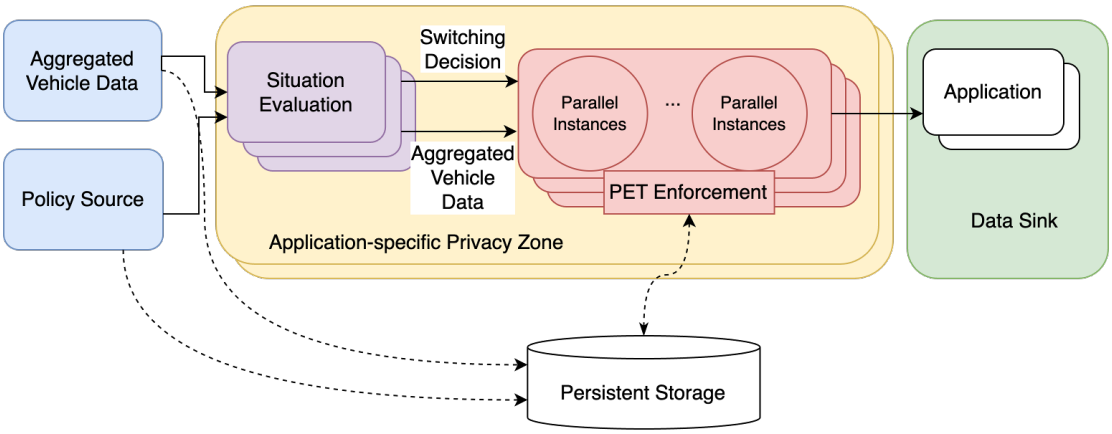


Figure 3.15: Architecture of the Framework with Refinement

in Figure 3.15, *AdaPrivFlow* supports the live adaptation of PETs of three modes proposed in Section 3.4.1: *Stateless Centralized PET Enforcement*, *Stateful Centralized PET Enforcement* and *Stateful Distributed PET Enforcement*,

4 Proof-of-Concept

In this chapter, the details of the practical aspect of *AdaPrivFlow* is introduced. In Chapter 3, we have designed the architecture of *AdaPrivFlow* and proposed the approach of live adaptation of PETs regarding the requirements in Section 3.1.1. Its feasibility of reaching the live adaption of PETs should be proved through implementation. For implementation, there exists various choice of popular technologies to realize the functionalities. In Section 4.1, we choose the underlying technologies by analyzing their features and if these are advantageous for achieving our goals. In Section 4.2, we build the connection of the chosen underlying technologies with our proposed architecture in Figure 3.15. Besides the choice of underlying technologies, other crucial parts in the implementation of *AdaPrivFlow* are covered in Section 4.3. The interface and the annotation of the modularized PET, as well as the format of the privacy policy will be introduced.

4.1 Underlying Technologies

In this section, we introduce the applied underlying technologies that support the functionalities of the major components in Figure 3.15. *AdaPrivFlow* utilized stream processing technology to meet the requirement R1. Therefore, in this section, we analyze the alternatives of the stream processing framework and the database. Through the comparison of pros and cons of each alternative, the choice is made.

Stream processing platform

We have compared the popular stream processing platforms Apache Flink, Apache Storm and Kafka Stream. We examined the processing mode, the functionality and flexibility provided by API, the deployment mode, the supported data source, the processing semantics and learning cost of the technology. The result of the comparison is listed in Table 4.1. According to the result, Apache Flink has the following advantages if employed in our proof-of-concept:

- *High flexibility*. Apache Flink stands out with its power API for streams. It provides comprehensive operation on the streams and state management. In the APIs from other platforms, part of the existing functionalities might need to be implemented manually, which introduces extra overhead.
- *Exactly-once semantics in nature*. With Apache Flink, the exactly-once processing semantics can be guaranteed without additional development efforts. Recall that PETs employ methods to protect privacy without sacrificing the data utility. Undesired loss or duplication of data will introduce negative effect on the utility. With the application of exactly-once processing semantics, the dataset enforced with PETs correspond exactly to the original dataset, ensuring the utility of the processed data not being harmed.

Table 4.1: Comparison of Popular Stream Processing Platforms [The23a][Con23][The23b]

	Apache Flink	Apache Storm	Kafka Stream
Stream	unbounded bounded batch	unbounded	unbounded
API	Multilayered API	Low-Level API	Multilayered API
Deployment	cluster	cluster	library
Data Source	Multiple source	Multiple source	Only Kafka topics
Processing Semantics	Exactly-once	At-least-once	At-least-once exactly-once
Accessibility	Intuitive Documentation	Steeper learn curve	Easy if stays in Kafka ecosystem

- *Outstanding potential of extendability*. Considering the future evolution of *AdaPrivFlow*, including incorporating Complex Event Processing (CEP) to enhance the evaluation of situation, as well as further integration with persistent storage, Apache Flink provides native CEP engine and native support for RocksDB as persistent storage backend.
- *Friendly learning curve*. Apache Flink has intuitive documentations and sufficient resources. On the contrary, according to multiple online information sources, Apache Storm has steeper learning curves. Besides, the user-friendliness of Kafka stream is conditional. It would be easy to be mastered if the developer is already immersed in the ecosystem of Apache Kafka.

Therefore, we leverage Apache Flink as the underlying stream processing platform, so that we can focus on the development of the core functionalities of *AdaPrivFlow* without introducing overheads from other technical issues.

Persistent storage

As is described in Section 3.4.1, a component for persistent storage is integrated in the framework. The main motivation is to store the history vehicle data, not only due to the necessity of meeting the regulation of law enforcement [LHSM23], but also with the aim of aiding the state building process for stateful PETs. Besides, it is also mentioned that the storage privacy policy and the local algorithm repository can also utilize the storage, making it a persistent storage component with general purpose.

A common suitable storage model for the history data, the privacy policy and the PET algorithm is the key-value storage. For history data, the key could be the data section name plus timestamp and the value is the sensor reading. For privacy policy, the key could be selected as the application name plus the timestamp. As described in Section 4.3.3, the privacy policy is application-specific. The PET algorithms are distinguished by their names, which is suitable to be the storage key.

Another approach is to utilize the relational database and store the data in distinct tables. It makes sense for history data. Through merging the tables, the relationship among the data sections can be established by linking the timestamps. A potential scenario of this usage is to reconstruct the real-time status of the vehicle during a certain period of time in the past. However, for privacy

policy and PET algorithms, relational storage mode doesn't provide any advantage. Therefore, in *AdaPrivFlow*, we utilize the key-value storage mode, so that the goal of integrating a persistent storage can be achieved by a unified solution.

Apparently, database access is not an efficient operation, compared with in-memory data access, such as by deploying an in-memory cache database like Redis [Red23]. Considering the proposed aim of backup, a persistent storage is indispensable. Moreover, in-memory cache database are usually used in conjunction with persistent storage databases,which increases the complexity of the whole framework. From another perspective, reading from the database occurs infrequently. To lower the complexity of *AdaPrivFlow* within this work, we do not employ caching component in the part of data backup.

For ease of implementation, we select the document-oriented database MongoDB [Mon23] as the persistent storage, which is inherently a subclass of key-value storage database.

4.2 Technological Implementation of Architecture

In Section 4.1, the underlying technologies for realizing the proof-of-concept are selected. The next step is to integrate them into the architecture in Figure 3.15. The merging result is shown in Figure 4.1. First, the mapping of the technologies to each architecture component is illustrated in Section 4.2.1. Then, the actual data flow with the application of technologies is described in Section 4.2.2.

4.2.1 Mapping of Technologies

As is described in Section 4.1, Apache Flink has been chosen to be the underlying stream processing platform. MongoDB has been selected to realize the persistent storage. In addition to Section 4.1, Apache Kafka is also applied in the implementation. The reason will be illustrated at the end of this section. In the following, the relationship between Figure 3.15 and Figure 4.1 is illustrated.

The 4 data pipeline components in Figure 3.15, namely, the input sources, the `Situation Evaluation` operator, the `PET Enforcement` operator and the `Data Sink`, can be implemented with Apache Flink. The sources in Figure 3.15 corresponds to the blue circles at the leftmost position in Figure 4.1. Apache Flink provide the functionality called `Source Function`, through which the input data can be programmatically generated and emit directly into the pipeline. Thus, the simulated data don't need to be transmitted via a message broker, before being ingested by the framework, reducing the implementation complexity.

The component `Situation Evaluation` is colored purple in Figure 3.15 and Figure 4.1. In the implementation, the *keyed* processing of Flink is employed. With a specific *key*, the states of processing are scoped to it [The23e]. This implementation choice has the following reason. This feature can be applied to the following fact during the assessment of situations. As is mentioned in Chapter 1, *AdaPrivFlow* is based on the previous work of Li et al. [LHSM23], where the privacy policy is defined with priorities. We have extended the privacy policy to suit our scenario of situational privacy protection. Details about this point is illustrated in Section 4.3.3. Consider a policy regarding data section `A`, `B` and `C`, it could be the case that the highest priority of the satisfied

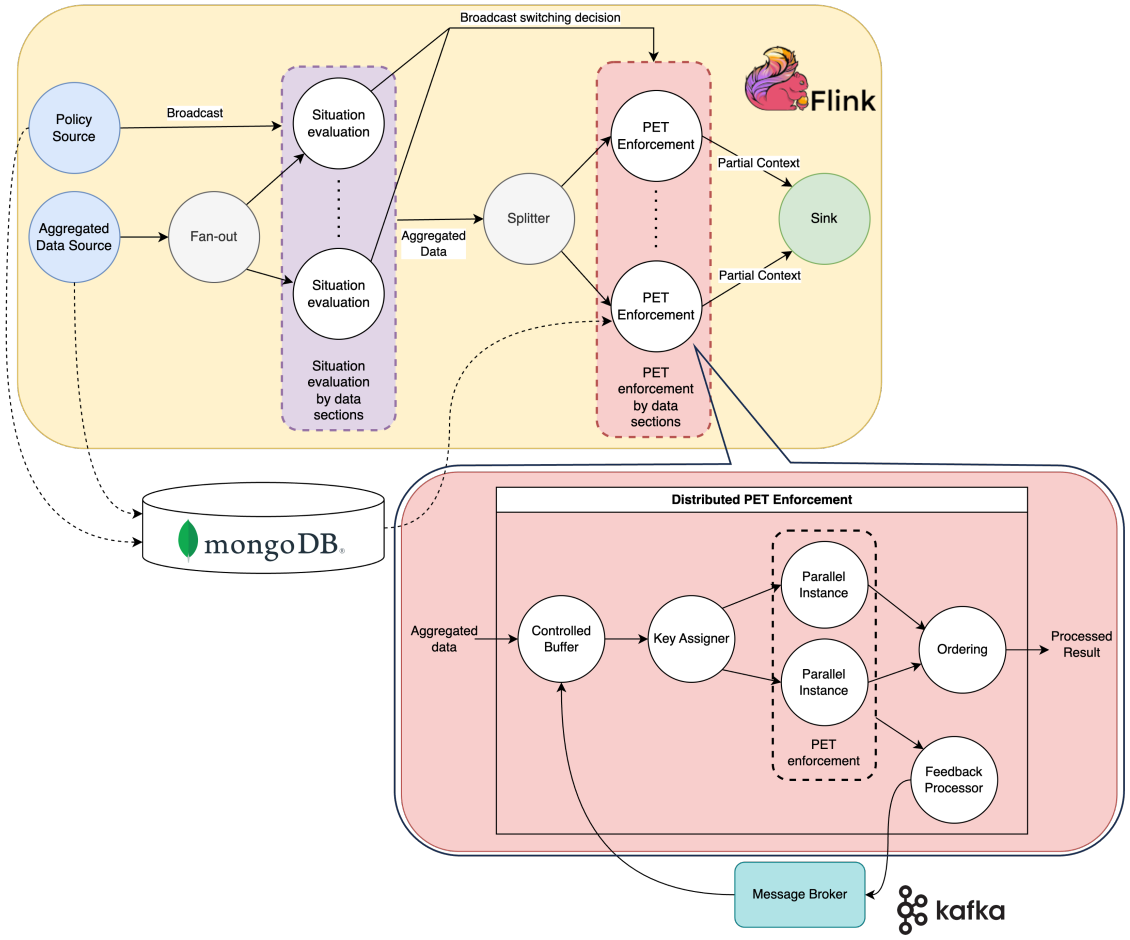


Figure 4.1: Implementation of the Architecture

conditions in each data section may be different. Thus, the operator should maintain a state, where the highest satisfied priority of each data section is documented. The necessity of documenting the highest satisfied priority lies in the argument that higher priority overrides lower priority. Take the scenario E1 in Chapter 1 as an example, the policy could be defined as utilizing PET *A* under all conditions with priority 5 and deactivate the PET by car accident with priority 4. In this example, numerically less number indicates higher priority. From this example, the necessity of noting the highest satisfied priority is explained. With the *keyed* processing functionality of Flink, the data section name could be set as the *key*. In this way, the maintenance of the highest satisfied priority regarding the data section could be easily implemented with the *keyed* processing.

Not to forget that between the sources and the `Situation Evaluation` component, an additional `fan-out` operator is inserted. This operator multiplies the aggregated vehicle data and practically tags each multiplied record with a distinct *key*, which is the name of the data section, against which it should be evaluated. If the data record were not fanned out, it could be only attached with one *key*, keeping the state of only one data section. The processing state of other data sections cannot be maintained without extra effort.

The component PET Enforcement are colored red in Figure 3.15 and Figure 4.1. Recall Figure 3.4, there are dedicated PET Enforcement operators processing each involved data section, each of which receives the switching decision and the aggregated vehicle data from the Situation Evaluator of the same data section. In Figure 4.1, this connection is realized with the help of a helper operator Splitter. Practically, the output of the Situation Evaluators are unified in one stream. The Splitter filters the output from each data section and relay them to the PET Enforcement operator of the same data section.

In Section 3.4.3, we proposed the *stateless distributed PET enforcement*, whose pipeline is shown conceptually shown in Figure 3.14, ensuring the fulfillment of requirement R3 and R5 for high dimensional vehicle data. This specific pipeline is shown in the dialog bubble, pointing to one PET Enforcement operator. One obvious deviation between the implementation and concept can be observed through comparing Figure 3.14 and Figure 4.1. In the conceptual pipeline in Figure 3.14, the feedback signals are directly sent back to the upstream operator Controlled Buffer. Due to the limitation of Apache Flink, that the stream topology are DAG [The22], the feedback can only be routed via an external message broker. Therefore, Apache Kafka is applied for this purpose. Since this using scenario is very simple, there is no preference or restrictions of choice. Other message broker can also be applied in this situation. Additionally, the feedback signals are not directly sent by each instance of *PET Enforcement* operators to the message broker, which is also due to the functional limitation of Flink. Thus, a Feedback Processor, which collects the feedback signals, is deployed as a workaround.

The last step of data presentation is implemented by the Sink Function provided by Flink. Through Sink Function, it could be simulated that the processed results is queried by an application involved in the privacy policy. For the purpose of demonstration, we implemented a UI for data visualization.

4.2.2 Data flow

Having illustrated the mapping relation between the conceptual architecture in Figure 3.15 and the architecture with applied technologies in Figure 4.1, We will walk through the data flow in Figure 4.1.

As is described in Section 4.2, the sources are implemented with Source Function provided by Flink. Beside generating and emitting privacy policies and aggregated vehicle data, respectively they also write the generated elements in MongoDB.

While the aggregated vehicle data is multiplied by the Fan-out operator and dispatched to the data-section-specific Situation Evaluators, the generated privacy policy is broadcast to all the Situation Evaluators. Through broadcast, each Situation Evaluator receives the same policy. The broadcast propagation eases the implementation. This simplicity is also leveraged during the propagation of the switching decision. They are also transmitted to the *PET Enforcement* operators via broadcast. The *PET Enforcement* operators identify whether the switching decision is related to their own data section. Meanwhile, as is described in Section 4.2, the multiplied evaluated aggregated data flow through the Splitter and filtered by the tag brought by the fan-out operator. Hence, each PET Enforcement operator receives one evaluated aggregated vehicle data. As the last step, the processed results flow into the Data Sink.

4.3 Miscellaneous Practical Aspects

As is pointed out in Section 3.3.1, the adaptation approach is set to *dynamic PET loading*, which requires the compatible PETs in *AdaPrivFlow* to be implemented in a modularized manner. Practically, the PETs should implement the same interface. In this section, we will first take a closer look at the interface design in Section 4.3.1. Following this, the annotation of the PETs is introduced in Section 4.3.2, since the `PET Enforcement` operator instantiates the PETs based on the annotation, as is shown in Figure 3.7. After the introduction the technical details about the PETs, the details about the definition of privacy policy will be introduced in Section 4.3.3, which serves as the guideline of adaptation.

4.3.1 Interface for Modularized PETs

In this section, we illustrate the concept of designing the interface for compatible modularized PETs. Figure 4.2 shows the UML diagram of the interface. Recall in Section 3.4.1, considering the reliance of history data records, the PETs are categorized into *stateful* and *stateless* PETs. In *AdaPrivFlow*, these two variants share the same interface definition. The interface has 4 methods, two of which are compulsory for both categories of PETs. Each PET should implement the `execute` method, which processes the input vehicle data. They should further implement the method `isReady`. This method is called by the framework to check if the warm-up phase is completed, for example in Algorithm 3.3 line 9 and in Algorithm 3.4 line 3. As for stateless PETs, this method returns unconditionally `true`. Additionally, stateful algorithms should implement the methods `buildState`. This method is overloaded intentionally with two types of input. It can either take a single aggregated vehicle data, or a list of them as input. The invocation of the former method occurs when a stateful PET is not sufficiently warmed up after the initialization. It should consume the arriving aggregated data to further build its internal state. Typical invocations can be found in Algorithm 3.3 line 10 and in Algorithm 3.4 line 4. The latter method is invoked during the instantiation phase (Algorithm 3.3 line 5). The queried result from the storage is a key-value pair, where the key indicates the data section and the value contains a list of history data. This data structure is designed with the consideration of the potential necessity that the state building using multiple data sections.

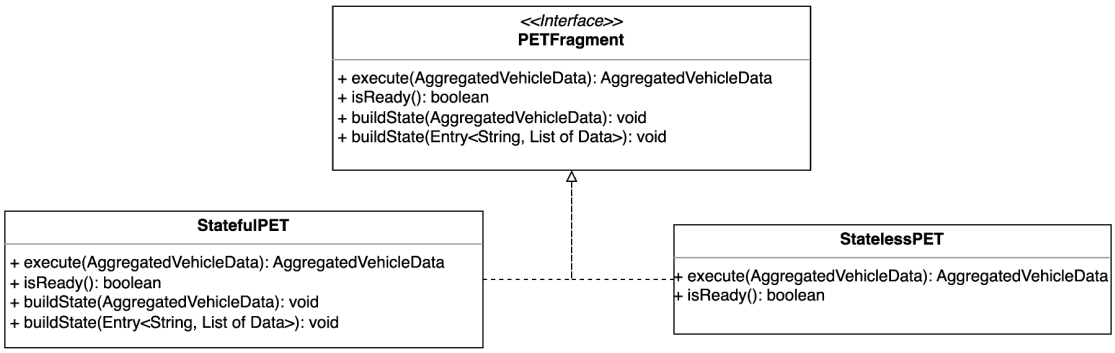


Figure 4.2: Interface Design of Modularized PETs

4.3.2 Annotation of PETs

In this section, the annotation of the compatible PETs is introduced. As is mentioned in Section 3.3.2, the annotation of PET is contained in the `switching decision`. Recall the depicted process of PET provision in Figure 3.7, after the `Switching Decision` is ingested by the PET enforcement component, it is parsed by the `Annotation Processor`. The `Annotation Processor` extracts the annotation and hand the parsing result to the `Algorithm Provider`. The `Algorithm Provider` finds, instantiates and initializes the PET based on the parsing result. The annotation is machine-readable and represented using JSON format. Listing 4.1 provides the annotation of a typical PET algorithm. In this example, the annotation of a dummy PET is shown.

The annotation contains the following information about the PET: its class name, the data section and the value type of its I/O, its dependency on history data and the parameter for initialization. The class name of the PET is given in `name`. The data sections, from which the PET get the necessary input, are stored under `inputDataSection`. In `outputDataSection`, the information about the data section(s) in the outcome of a PET is given. These two field contains the indispensable information for the adaptation approach based on a dynamic pipeline, like the Variant 2 in Figure 3.6, since the rebuild of pipeline needs the I/O information about each PET operator. These two fields are reserved for future use. The fields `inputType` and `outputType` represent the class types of the I/O. They are also necessary for dynamic pipelines, because the ingested vehicle data can be aggregated on purpose with dynamic pipelines, instead of containing all the data sections and forming an instance of `Vehicle Context`, as is given in the assumptions in Section 3.1.2. These fields are also reserved for future use. The information about the reliance on history data records of *stateful* PETs are given in `stateWindow`. This name is inspired by the concept *windowing* in stream processing,

Listing 4.1 The Annotation of An Exemplary PET Algorithm

```
{
  "name": "thesis.pet.repo.AverageSpeed",
  "inputDataSection": ["speed"],
  "outputDataSection": ["speed"],
  "outputType": "java.lang.Double",
  "inputType": "java.lang.Double",
  "stateWindow": [
    {
      "dataSection": "speed",
      "type": "piece",
      "size": 5
    }],
  "parameters": [
    {
      "name": "size",
      "type": "java.lang.Integer",
      "value": 5
    }
  ]
}
```

where the stream is split into segments of finite size [The23d]. A `stateWindow` defines the data section the records belong to and the amount or the time span of the necessary history data. In *AdaPrivFlow*, two categories of windows are supported:

1. *Time-based window*. It is declared by setting the value of `type` to `time`. It defines a window's size measured by an interval. For instance, a PET relies on the records from the last hour. The length of the interval is given in the unit of seconds. The return of the database query with this kind can contain arbitrary number of records, including 0.
2. *Piece-based window*. It is declared by setting the value of `type` to `piece`. This type of window defines the total number of history results. A query in the data storage of this kind contains maximal the given number of results. The query result has fewer records if not enough data exist in the storage. In this case, the PET will be warmed up with the incoming aggregated vehicle data without producing any output, until the necessary number of data is reached. Line 3 to line 7 in Algorithm 3.4 describe the action under this condition.

A `stateWindow` can contain multiple triples consisting of `dataSection`, `type` and `size`, since a stateful PET can rely on history data from multiple data sections. As for the example in Listing 4.1, we could clear read that this dummy PET relies only on `speed` and needs 5 history records to produce the result. The value of `parameters` is a list of parameters, which are used to instantiate the PET algorithm. Not only the value, but also the classes are given, since they are necessary for applying the parameters in the correct constructors.

The annotation of the dummy PET in Listing 4.1 participates the PET switching in Figure 3.7 in the following way. The `Algorithm Provider` receives the parsing result of the annotation from the `Annotation Processor`. It finds the PET by the `name` in the local algorithm repository and instantiate it with the `parameters`. In the switching step, Algorithm 3.3 is called. Based on the information in `stateWindow`, the database query takes place and the returns result is an `Entry`, whose key is the `speed` and whose value is a list of 0 to 5 speed readings, depending on how many history data are available. This `Entry` is the input argument for the method `buildState` in Figure 4.2. In this way, the switching procedure is finished.

4.3.3 Privacy Policy Format and Evaluation

As is mentioned in Section 3.2.1, the focus of this work lies in the mechanism of adaptive switching, which is the procedure happened inside the `PET Enforcement` component (ref. Figure 3.15). For the purpose of evaluating the framework in a nearly real environment as well as demonstration, we also implemented the components prior to the `PET Enforcement` in Figure 3.15. We have proposed an extended format of privacy policy and the `Situation Evaluator` suitable for this format. In the following, we will describe the details about the foundation of adaptation: the privacy policy.

The previous work of Li et al. [LHSM23] has proposed the definition of a privacy policy. In this definition, the values of the following four fields are crucial for evaluating the situation as well as enforcing the PETs: (a) the applicable conditions, (b) the priority, (c) the target services, and (d) the target data sections. In their work, the proposed policy definition is given under one condition. Take the border crossing scenario in Chapter 1 as an example. The policy in their work would be defined as utilizing PET *A* in Europe. However, this definition doesn't distinguish the difference of privacy policies in each country. If the nations sharing a boarder has different regulation, this

Table 4.2: Complete Definition of the Privacy Policy in the Form of a Table

app_name: my_app

	Priority *x*		Priority *y*		Priority ...
Data section *A*	condition 1	PET 1	condition 2	PET 2	
	...	...	...	...	
Data section *B*	condition 3	PET 3	condition 4	PET 4	
	...	...	...	...	
...					

Listing 4.2 The Structure of A Completely Defined Privacy Policy

```
- Policy
    -- app_name
    +- rules
        +- single_data_section_rule
        +- ...
        +- single_data_section_rule
            -- data_section
            +- atomic_rule
            +- ...
            +- atomic_rule
                -- priority
                -- condition
                -- PET_annotation
```

policy definition cannot express this difference. In order to lively adjust to the difference, the policy definition should be extended. The policy for the mentioned example could be written as PET *A* in EU-countries and PET *B* in Switzerland. We therefore propose an appropriate format of policy for *AdaPrivFlow* to deal with this necessity.

In our proposal, we inherit the terms in the policy definition in the previous work, except that `service` is replaced by `application`. Our proposed privacy policy can be expressed as a table, as is shown in Table 4.2. The table can be decomposed into the structure in Listing 4.2. Here an exemplary privacy policy of the application named `my_app` is presented. It queries the information from `Data Section A`, `B`, and so on.

Now let's look at the relationship between Table 4.2 and Listing 4.2 from the point of view of content. On the top level in Listing 4.2, the policy consists of two fields: the application name `app_name` and its `rules`. The `rules` are the contents in the table in Table 4.2. This design choice allows the individual definition of rules for each querying app. Recall the requirement R2, the privacy protection should be application-specific. With the individual `rules`, this requirement can be satisfied. The `rules` can be expanded into multiple `single_data_section_rules`. They are defined for each involved data section, which correspond to the row of those data sections in Table 4.2. Besides the specification of the data section, one single `single_data_section_rule` consists a set of *atomic_rule*s. The *atomic_rule*s are the basic elements of a complete privacy policy. They have three attributes: `priority`, `condition`, and `PET_annotation`. The information about the annotation of PETs is introduced in Section 4.3.2. Apparently, if a PET is applicable, its affiliating `condition` must

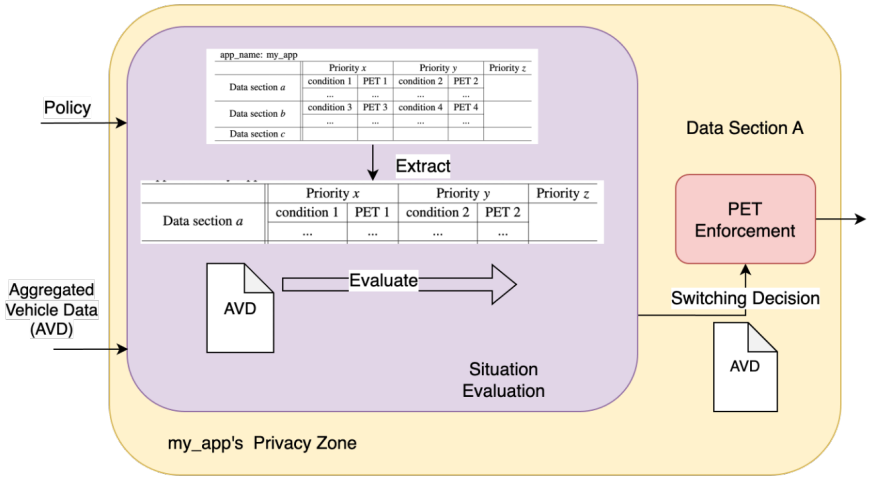


Figure 4.3: Decomposition of the Privacy Policy inside the Situation Evaluation Component

be assessed as `true`. Multiple *atomic_rule*s can share the same value of `priority`. An *atomic_rule* can be derived in Table 4.2 through (1) extracting the row of the data section it belongs to, and then (2) extracting the column of its `priority`. Each row in the result is then an *atomic_rule*. For a rule for a single data section, the atomic rules will be evaluated from high priority to low priority. The privacy policy can be stored in a machine-readable form in a persistent storage. In *AdaPrivFlow*, the JSON format is applied to store the complex structure of a complete policy.

Figure 4.3 shows how the pipeline in Figure 3.4 is able to evaluate an aggregated vehicle data against the policy definition for each data section simultaneously. In this figure, we zoom out the individual pipeline for the app `my_app` and focus on the situation evaluation component. The dedicated `Situation Evaluator` for `Data Section A` extracts the `single_data_section_rule` for `A` and stores it as its internal state. Every time an aggregated vehicle data record is ingested, it will be evaluated against the *atomic_rule*s from high priority to low priority. For other data sections, the approach is identical. They are omitted in Figure 4.3. In this way, the dedicated evaluators for each queried data section operate simultaneously.

5 Evaluation

In Section 3.1.1, five requirements of realizing live adaptation of PETs in the data pipeline of a CV are outlined, which serve as the foundation for the evaluation. In this section, we will evaluate the proof-of-concept in Figure 4.1 against these requirements. We will first analyze the fulfillment of each requirement and derive the evaluation methods in Section 5.1. Then, through the introduction of the experiments and their results, further in-depth evaluation of *AdaPrivFlow* will be presented in Section 5.3. The learned lessons from the experimental results the discussion about the limitations will be illustrated in Section 5.4.

5.1 Fulfillment of Requirements

The requirements proposed in Section 3.1.1 can be divided into two categories: *non-quantifiable* and *quantifiable* requirements. While the requirement R1 to R4 belong to the former category, R5 falls into the category of the latter.

The non-quantifiable requirements are examined first. The requirement R1 is supported by two main elements. Firstly, The automatic reaction to situational change guaranteed through the underlying stream processing technology. Several important requirements for stream processing technology have been outlined by [SÇZ05]. Those related to our specific framework are keeping the data moving, generation of predictable outcomes, and process and respond instantaneously. These features are implemented by popular stream processing frameworks. By leveraging these frameworks, the input are ingested and processed on time. Specifically in our framework, as shown in Figure 3.15, the aggregated vehicle data and the privacy policy can be promptly processed. The aggregated vehicle data are continuously evaluated against the situation. The privacy policy is comprehensive and self-contained, not relying on interaction with the user. The decision is made based on the privacy policy input. Thus, the reaction of situational change is automatic and without user intervention. Secondly, the necessity of restart is avoided by the developed switching mechanism. Every time the framework reacts to situational changes and performs adaptation of PETs, the data pipeline still keeps running. Through the steps of annotation parsing and instantiation, as depicted in Figure 3.7, the next PET is prepared. With the feasible condition of switching in Section 3.3.3 and its refinements in Section 3.4.2 and Section 3.4.3, the switching of next PET is completed. Therefore, the adaptation can be completed during the runtime. The whole system doesn't need to be restarted.

The requirement R2 is fulfilled through the format of the privacy policy and the design of data pipeline. In Section 4.3.3, we proposed the extended privacy policy format based on the previous work of Li et al. [LHSM23]. The privacy policy is application-specific, thus enabling the individual definition of policies. As is shown in Figure 3.3, each querying application is equipped with its

individual pipeline, sharing the input sources of privacy policy and aggregated vehicle data. This design enables the individual evaluation of the situation and enforcement of the PETs according to the individual privacy policy and enforce the PET.

The requirement R3 is satisfied primarily through the feasible condition of switching, which is illustrated in detail in Section 3.3.3. The assessment of this condition is based on the identification of the *trigger* and the processing of switching decision. Once the switching decision is processed, the next PET contained is prepared, as is shown in Figure 3.7. The next PET will be activated only if the affiliating *trigger* is ingested. Thus, the correct enforcement of PET on the vehicle data is ensured. In Section 3.4.3, this feasible condition is extended in the distributed PET enforcement, which is designed to cope with high dimensional data and long-running process. In this way, the requirement R3 is satisfied considering common vehicle data type in a CV and PETs processing these data.

The requirement R4 is ensured through the definition of the interface and the annotation for the PETs to be applied in *AdaPrivFlow*. In addition, the integration of a database in this framework also contributes to the guarantee of this requirement. Recall that in our design choice, Variant 3 in Figure 3.6, the PETs share the `PET Enforcement` operator. This requires a modularized design manner, so that the PETs could be switched easily. In Section 4.3.1, we proposed the interface of the modularized PETs. The characteristics of *stateful* PETs compared with the *stateless* ones are reflected in the design of interface and annotation. The annotation enables the correct parsing of the reliance on history data. The interface provides the method to help the stateful PETs build their internal state. The integration of the data provides the possibility of storing the necessary data persistently.

The requirement R5 is quantifiable. In Section 2.1, the diversity of vehicle data on a CV is introduced, featuring with varying data volume and refresh rate. As is described in Section 3.4.1, low processing latency is expected in some application scenarios, especially the processing of camera data, which is characterized by its large volume and refresh rate. Therefore, the fulfillment of R5 can be evaluated by the *end-to-end latency* of the vehicle data. This value indicates the elapsed time from the data entering the framework till its readiness of being queried by the applications. It should be investigated, to which extent of the common data types in a CV the requirement R5 can be ensured.

In the following part of this chapter, the evaluation method of R5 is introduced. In the evaluation, both a theoretical examination of the latency sources and experimental results are considered. The aim of the evaluation is to explore the impact of various strategies in Chapter 3 for the live adaptation of PETs on the end-to-end latency of the aggregated vehicle data. In Section 5.2 the origins of latency are analyzed. The experiment plan and result can be found in Section 5.3. An analysis of the potential limitation revealed during the evaluation and the optimization possibilities regarding the performance are discussed in Section 5.4.

5.2 Theoretical Analysis of the Origins of Latency

As is described in Section 5.1, the requirement R5 proposed in Section 3.1.1 could be evaluated quantitatively by measuring the data's end-to-end latency. Considering the data flow shown in Figure 4.1, a record of aggregated vehicle data undergoes the processing multiple processors, such as

those for `Situation Evaluator` and `PET Enforcement`. Since the end-to-end latency is the summation of the processing latencies of each operator plus the transmission time from one operator to another, the behavior of the operators have impact on the end-to-end latency. Therefore, it's necessary to measure the latency with finer granularity. Furthermore, the *trigger* undergoes buffering during the adaptation process of PETs, as is described in the feasible condition of switching in Section 3.3.3 and its refinement with multiple parallel instances (Section 3.4.3). This approach not only increases the value of the end-to-end latency, but also is strongly dependent on the processing of switching decisions. To address the issue, our focus lies in the fine measurement of the *trigger* data and its affiliating switching decision.

In order to perform a fine measurement, the preliminary step is to establish the key measuring points. In the following, we identify several critical timestamps to monitor and analyze the journey of the *trigger* and its relevant switching decision process. Each timestamp captures a specific phase in this progression.

- *Generation Timestamp (t_g).* t_g is inserted when a data record is produced by the `Source Function` of Apache Flink. Comparing with Figure 4.1, it is added just before a data flows out of the sources colored in light blue. The time of generation is equivalent to the time when this data record is supposed to enter the *AdaPrivFlow*. It serves as the starting point of the latency measurement.
- *Decision Timestamp (t_d).* This timestamp denotes the end of the situation evaluation phase. It is the time when this data record is ready to be transmitted to the operators for PET enforcement. In Figure 4.1, it is added just before the data flows out of the `Situation Evaluators`.
- *Processing Begin Timestamp (t_{p1}).* This timestamp reflects when a data record is about to be processed by the PET. It's worth noticing that this timestamp doesn't necessarily coincide with the entry into the `PET Enforcement` operator.
- *Processing End Timestamp (t_{p2}).* This timestamp marks the completion of the enforcement of a PET algorithm.
- *Presentation Timestamp (t_e).* A processed data record's readiness for the querying application is captured by this timestamp. For centrally enforced PETs, the timestamp is added before the processed data flows out of the `PET Enforcement Operator`. In case of distributed PET enforcement, the timestamp is added before the processed data flows out of the `Ordering` operator in Figure 4.1. At this time the order restoration is completed.

The process of switching decision has a direct influence on the switching of the PET algorithms. The following moments mark the important milestones during the processing of a switching decision:

- *Issuing Timestamp (t_u).* This timestamp marks the creation of the switching decision. A switching decision is emitted at the end of the situation evaluation, if a change in the evaluation exists. This timestamp aligns with t_d of the data record triggering this switching event.
- *Instantiation Begin Timestamp (t_{i1}).* This timestamp captures the onset of the instantiation of the next PET algorithm. Since the PET is instantiated immediately after the switching decision is processed by the `PET Enforcement` operator, as is shown in Figure 3.7, this timestamp is added the moment the switching decision is ingested by the `PET Enforcement` operator.

- *Instantiation End Timestamp (t_{i2}).* This timestamp denotes the successful processing of the switching information and instantiation of the PET algorithm.
- *Switching Begin Timestamp (t_{s1}).* As the name suggests, it marks the start of the transition of the PET from a pending to active state. It's worth noticing that the onset of the switching process doesn't always immediately follow a successful instantiation. As proposed in Section 3.3.3, the switching procedure takes place only after the *trigger* for this switching event is acknowledged by the PET enforcement operator. Thus, a time gap might emerge between the switching and instantiation stages, if the *trigger* is delayed.
- *Switching End Timestamp (t_{s2}).* This timestamp marks the completion of the switching phase.
- *Synchronization Timestamp (t_{sync}).* This timestamp is particularly relevant for the switching decisions of distributed PET enforcement. As described in Figure 3.14, each parallel instance of the PET enforcement operator emits a `READY` signal upon successful instantiation of the new PET. This timestamp is added by `Feedback Processor` in Figure 4.1 upon receiving the final `READY` signal, indicating the end of the switching procedure among the parallel instances.

From the measurement points listed above, we could compute the latency components of the *trigger* latency and switching decision latency. The latency components are listed in Table 5.1. In Table 5.1a, the latency components of the *trigger*. The evaluation latency is the elapsed time from data generation to the end of situation evaluation. The transmission latency indicates specifically the transmission from `Situation Evaluation` to `PET Enforcement`. The processing latency measure the computational time of the PET algorithm. The presentation latency is only valid in distributed PET enforcement, because the data under centralized PET enforcement is ready to be supplied to the querying application once the PET processing is finished. As is already described in Section 5.1, the end-to-end latency measures the elapsed time from the data entering the framework till its readiness of being queried by the applications.

The latency components of the switching decision are listed in Table 5.1b. The transmission latency has the same definition as the one of the *trigger*. The instantiation latency measures the duration of PET instantiation. The switching latency measures the elapsed time of the PET switched from the pending state to working state (Figure 3.7). Recall Algorithm 3.3, it's noteworthy that the time expenditure of database access by stateful PETs is contained this value. The duration of database

Table 5.1: Latency Component of Trigger Data and Switching Decision Represented with Measured Timestamps

(a) Trigger

Latency Component	Definition
Evaluation Latency (EV)	$t_d - t_g$
Transmission Latency (TR)	$t_{p1} - t_d$
Processing Latency (PR)	$t_{p2} - t_{p1}$
Presentation Latency (PS)	$t_e - t_{p2}$
End-to-end Latency	$t_e - t_g$

(b) Switching Decision

Latency Component	Definition
TR	$t_{i1} - t_u$
Instantiation Latency (IS)	$t_{i2} - t_{i1}$
Switching Latency (SW)	$t_{s2} - t_{s1}$
Synchronization Latency (SC)	$t_{sync} - t_d$

access is called State Building Latency (SB). The synchronization latency applies only to switching decisions regarding distributed PET enforcement. It gives the elapsed time from the decision initiation to the successful conclusion of the switch.

Having established the measurement plan, we determine the pipelines under investigation in the following. Recall Section 3.3.1, the approach of output filtering (Variant 1 in Figure 3.6a), dynamic pipeline (Variant 2 in Figure 3.6b) and dynamic PET loading (Variant 3 in Figure 3.6c) are compared. The design decision is determined to be Variant 3. This approach involves the provision of PETs and feasible switching condition, as is already introduced in Section 3.3.2 and Section 3.3.3. Notice that the switching procedure in Variant 1 involves only the adaptation of the filter criteria in the data sink. Through a comparison between Variant 1 and Variant 3, the impact of the increased complexity in the switching procedure from Variant 3 can be evaluated. Moreover, for a single data section, multiple PETs participates in the computation in Variant 1, while in Variant 3, only the current applicable PET is integrated in the pipeline. The comparison between these two variants can further reveal the impact of the number of concurrently running PETs on the latency of the aggregated vehicle data.

Recall in Section 3.4, the proposed adaptation procedure in Section 3.3 is optimized with the consideration of stateful PETs and high dimensional vehicle data. Regarding stateful PETs, the database access during the switching phase, as is described in Algorithm 3.3. It is already pointed out in Section 4.1 that database access is not a relative efficient operation. An experiment should be conducted to investigate the impact of database access on the latency of switching. As the baseline of comparison, Variant 1 will be compared with stateful centralized PET enforcement of Variant 3, since the stateful PETs keeps running in Variant 1, even if they are not applicable. Their internal state is maintained in the runtime. The switching procedure in this case involves only the adaptation of the filter criteria. Stateless centralized PET enforcement of Variant 3 is inappropriate to be chosen as the comparison baseline, since a stateful PET would be operating in cold start. The advantage of warm start through database access against cold start is trivial.

Regarding high dimensional vehicle data, distributed PET enforcement is introduced in Section 3.4.1. This approach introduces multiple parallel instances to process multiple aggregated vehicle data simultaneously, aiming at avoiding unbounded queuing of data in a single `PET Enforcement` operator due to long-running process. Meanwhile, to cope with unsynchronized switching and data disorder, corresponding measurements are taken, as is described in Section 3.4.3, increasing the complexity of the switching procedure. The positive impact from multiple parallel instances on the data latency and the potential negative impact from the switching procedure on the switching latency should be evaluated.

Table 5.2: Experiment Plan

Experiment Subjects	Experiment Purpose
Variant 1 vs. Variant 3 Stateless centralized	Impact of additional steps in PET loading Impact of number of concurrently running PETs
Variant 1 vs. Variant 3 Stateful centralized	Evaluate efficiency of database access
Variant 3: Stateless centralized vs Stateless distributed	Evaluate effectiveness of distributed PET enforcement Impact of synchronized switching and order restoration

To sum up, we will conduct 3 groups of experiments using the Variant 1 in Figure 3.6, the centralized PET enforcement of Variant 3, and the distributed enforcement for stateless PETs of Variant 3. The experiment plan and purpose described just now are listed in Table 5.2.

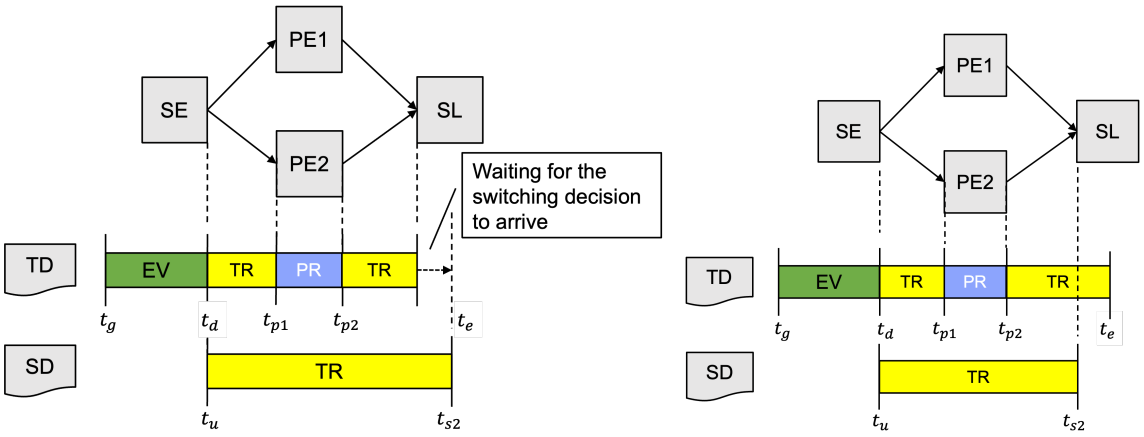
For the purpose of visualization, the measuring points are integrated in the data flow of each pipeline in Table 5.2, The results are shown from Figure 5.1 to Figure 5.3. In these figures, beside the abbreviations listed in Table 5.1 the following abbreviation of the Flink operators: Situation Evaluator (SE), PET Enforcement (PE), Controlled Buffer (CB), Selector (SL), Feedback Processor (FP), Key Assigner (KA) and Order Restore (OR). The *trigger* and the switching decision are written as Triggering Data (TD) and Switching Decision (SD), respectively.

Pipeline of Variant 1

Figure 5.1 describes the path of a *trigger* along with its corresponding switching decision in the benchmark pipeline. Uniquely in this approach, every candidate PETs is involved in the computation. This fact is depicted by two drawn PET enforcement operators in the figure.

The switching decision is routed directly to the selector positioned at the end of the pipeline. The selector is capable of excluding the results generated by presently non-relevant PETs. Similar to the introduced approaches in Section 3.3.3, the selector alters its filtering criteria only after it receives the *trigger*.

If the data that triggers the switching event reaches the selector prior to the switching decision, the data will be temporarily buffered as the selector awaits the switching decision. From the perspective of the selector, it is unknown what the subsequent action would be. As is illustrated in Figure 5.1a, the *trigger* must wait for the belated switching decision. Its presentation timestamp t_e is aligned with the completion of the switching process t_{s2}. On the contrary, if the *trigger* arriver later than the switching decision (Figure 5.1b), it could be presented immediately under the condition that it is processed with the correct PET.



(a) The trigger data arrives earlier than the switching decision

(b) The trigger data arrives later than the switching decision

Figure 5.1: Latency Measuring of Data and Switching Decision in Variant 1

The latency of the switching decision and the data records can be formulated as:

(5.1) $T_{SD} = \text{Trans}\{SE, SL\} = t_{s2} - t_u$

The latency of the *trigger* data is formulated as

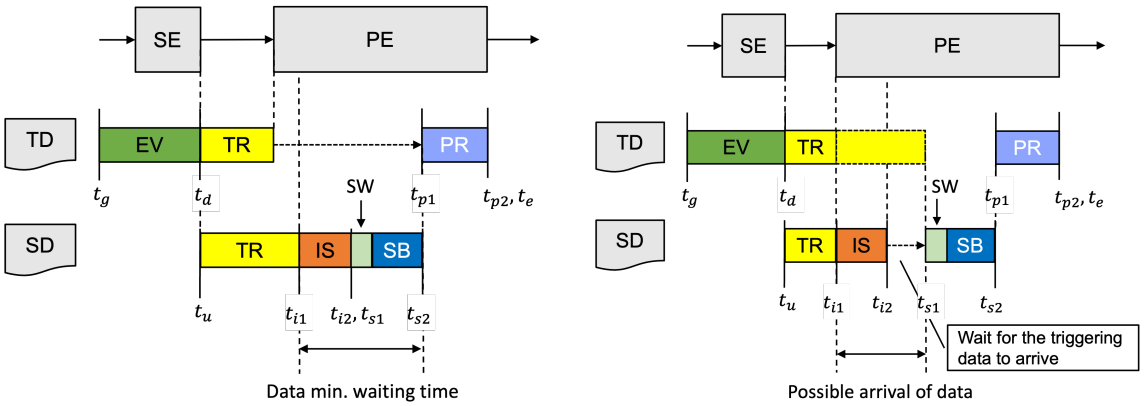
(5.2) $T_{TD} = T_{EV} + \max(\text{Trans}\{SE, PE\} + T_{PR} + \text{Trans}\{PE, SL\}, T_{SD})$

where $\text{Trans}\{A, B\}$ means the transmission time from operator A to operator B. This formulation is also used in the following analysis. As is shown in Equation (5.1), the latency of the switching decision is the transmission time from the situation evaluator to the selector at the end of the pipeline. Equation (5.2) conveys the information that the latency of the *trigger* is at least the latency of the switching decision.

Variant 3: Centralized PET Enforcement

Figure 5.2 illustrates temporal sequence of each phase of the switching procedure. The *trigger* undergoes evaluation in SE and PET enforcement in PE. The switching decision is emitted from the SE and processed by PE. Based on the arrival order of *trigger* and switching decision, the depiction is divided into two distinct scenarios. In this illustration, a stateful PET algorithm is selected as the end effect of this switching procedure for centralized PET enforcement. Should it be a stateless PET, the SB is excluded. For better visualization, the box of PE is elongated, aligning its right side with the moment the data record is ready to be released to the downstream. In this pipeline, the processed data record is ready to be queried immediately. Therefore, $t_e = t_{p2}$.

Figure 5.2a represents the scenario where the data record precedes the arrival of switching decision. In this case, it has to wait until the switching decision is processed by the PE. The shortest waiting interval occurs if the switching decision is sequentially processed immediately by the PE after the *trigger*. In this optimal situation, $\text{Trans}\{SE, PE\}_{data} \cong \text{Trans}\{SE, PE\}_{SD}$. Therefore, the minimum elapsed time of the *trigger* is approximately the complete processing duration of the switching decision.



(a) The trigger data arrives earlier than the switching decision

(b) The trigger data arrives later than the switching decision

Figure 5.2: Latency Measuring of Data and Switching Decision in Variant 3

Figure 5.2b illustrates the converse scenario, where the switching decision is processed earlier than the *trigger*. Here, the final switching step is postponed, because it has to wait for the appearance of the *trigger*. It is worth noticing that the time of waiting is not incorporated into calculating the overall processing time of switching decision. Therefore, in both cases, the composition of the switching decision latency remains the same. It is represented as the summation of the time spending for TR, IS and SW, and, when switching to stateful PET algorithms, additional time for database access. The elongated box of TR of the *trigger* indicates the potential time window of arriving at the PE. If the arrival falls in the instantiation phase of the next PET, i.e., between t_{i1} and t_{i2}, its processing will be deferred until the instantiation is finished, since an operator in the pipeline handles just one input at a time. Regardless of whether instantiation is blocking or asynchronous, the *trigger* can only be processed by the succeeding PET after the completion of entire switching procedure. As a result, the minimum duration of the *trigger* between evaluation and processing equals the duration of the switching procedure.

In conclusion, the latency of the switching decision and the data records for centralized PET enforcement pipeline can be mathematically described as follows:

(5.3) $T_{SD} = \mathsf{Trans}\{SE, PE\} + T_{IS} + T_{SW} + T_{SB}$

The latency of the *trigger* data is formulated as

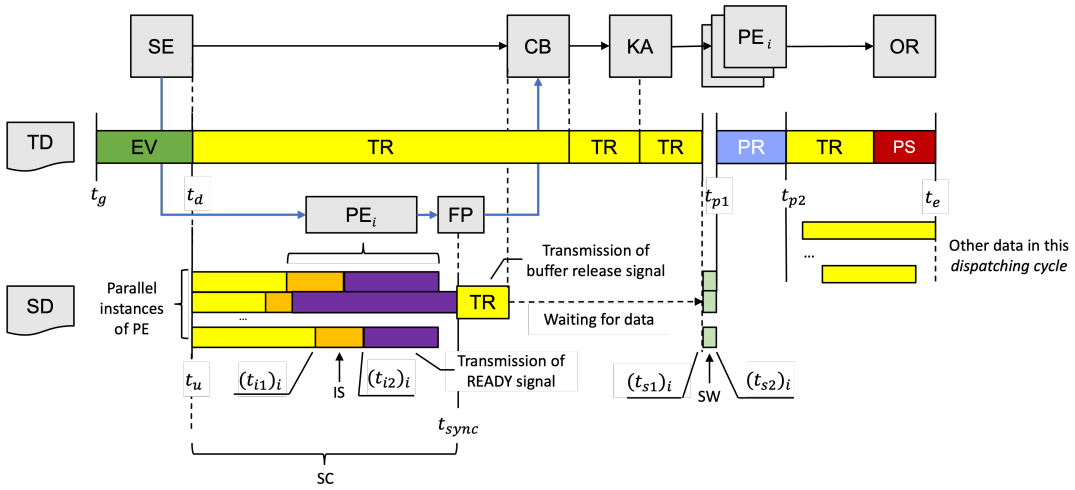
(5.4) $T_{TD} \geqslant T_{EV} + T_{SD} + T_{PR}$

By comparing Equation (5.1) with Equation (5.3), it can be noticed that the switching decisions in Variant 3 and Variant 1 are both transmitted once between the operators. The value of the term of transmission latency should be similar in both equations. However, due to the steps of instantiation, initialization and switching, three extra terms are added in Equation (5.3). As is shown in Equation (5.4), apart from the evaluation latency and PET processing latency, the latency of the *trigger* should at least wait for the processing of switching decision to be completed.

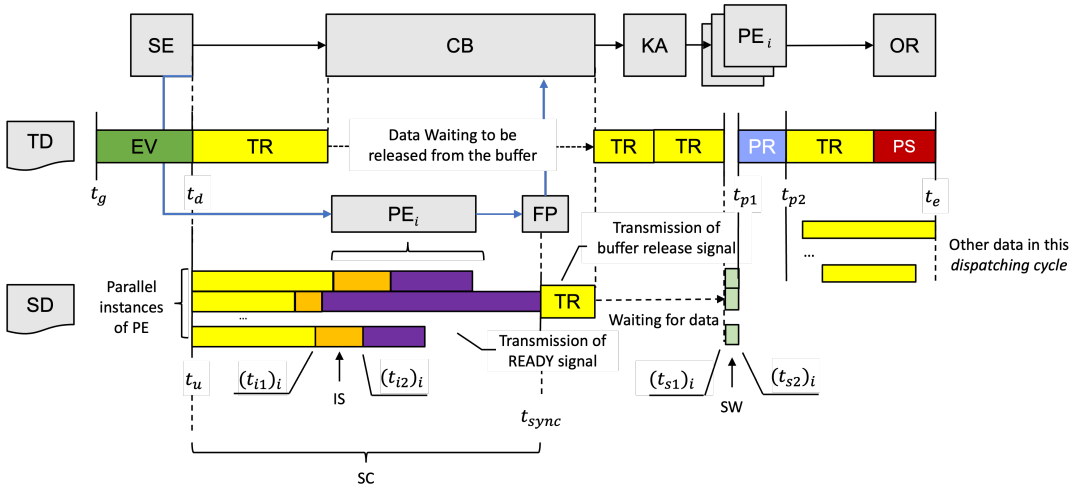
Variant 3: Distributed Stateless PET Enforcement

Figure 5.3 illustrates composition of latency in distributed PET enforcement. The pipeline in the figure is drawn based on Figure 4.1. Given the complexity of this approach, we use colors to distinguish the paths of the *trigger* (shown in black) and the switching decision (shown in blue).

Refer to Figure 4.1, each parallel instance receives the same the switching decision via broadcast. The lower part of Figure 5.3a and Figure 5.3b depicts the activities inside the parallel instances. Following the instantiation (IS) of the next PET and its activation (SW), each instance emits a `READY` signal to the `Feedback Processor`(FP). The traveling time of this signal to the `Feedback Processor` is denoted as SC. The time consumption of the switching decision to be transmitted to the parallel instances and the subsequent steps vary across the parallel instances, since it depends on the load and the scheduling of the CPU. Nevertheless, the duration of the switching procedure can be clearly defined using the decision time t_d and the *maximal* value of the synchronization time t_{sync}, since the system is only recognized as *ready* for enforcement with the subsequent PET after receiving the final `READY` signal from the parallel instances. Therefore, the time to achieve a synchronized state among these instances equals $t_{sync} - t_d$. Each instance of PE activates the new PET and proceeds with processing, once an aggregated data is detected, whose decision time t_d is the same as or later than the issuing time t_u of the switching decision.



(a) The trigger data arrives earlier than the switching decision



(b) The trigger data arrives later than the switching decision

Figure 5.3: Latency Measuring of Data and Switching Decision in Distributed PET Enforcement. The black line guides the flow path of the data. The blue line guides the flow path of the switching decision.

Referring the pipeline for data order preservation in Figure 3.11, each data record experiences an extra computational step of key assignment in `Key Assigner` (KA) prior to `PET Enforcement` (PE). After being enforced with the PET, the data is not ready for the external querying applications. The order of the data in this round must be reinstated, requiring all the data to arrive in the `Ordering` operator (OR). The presentation time t_e is thus determined by the latest arriving record in its *dispatching cycle* (ref. Section 3.4.3).

As a summary, the latency of a switching decision in distributed PET enforcement is

$$T_{SD} = T_{SC} + T_{SW} = t_{sync} - t_d + T_{SW} \tag{5.5}$$

The latency of the *trigger* data can be formulated as

(5.6)
$$T_d \geqslant T_{EV}+T_{SD}+T_{PR}+T_{PS}+\mathsf{Trans}\{FP, CB\}+\mathsf{Trans}\{CB, KA\}+\mathsf{Trans}\{KA, PE\}+\mathsf{Trans}\{PE, OR\}$$

As is shown in Equation (5.5), the switching decision latency consists of the synchronization latency and the switching latency. Compare with the latency of switching decision in centralized PET enforcement, there exists one additional term of `READY` signal transmission. Compare the data latency in Equation (5.6) with Equation (5.2) and Equation (5.4), the data is transferred much more times than in other cases. Moreover, in the case depicted in Figure 5.3a, where the *trigger* is temporarily buffered due to the delay of switching decision processing, since the release of the *trigger* occurs under the instruction from the `Feedback Processor` (FP), there's one more transmission step involved. In summary, the transmission between operators the major contributor to the end-to-end latency of the data.

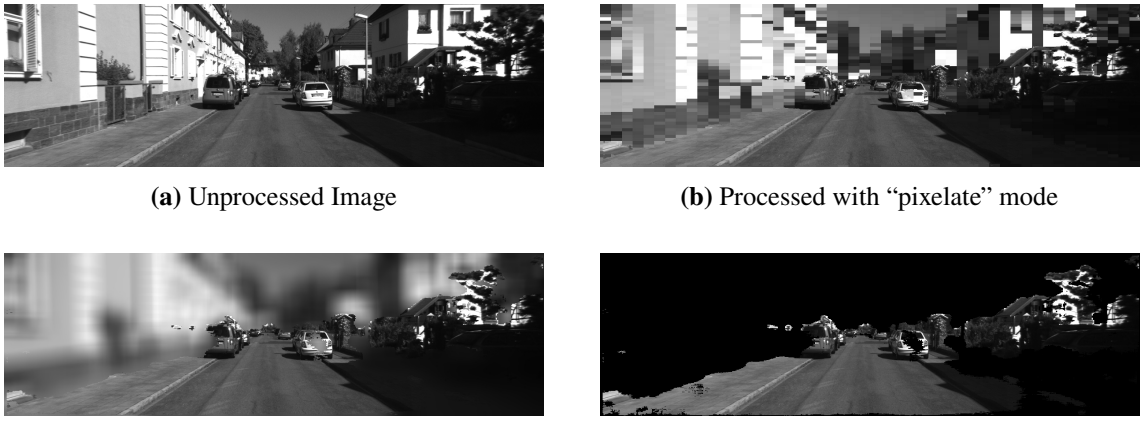
5.3 Experimental Evaluation

As is listed in Table 5.2, we conduct three set of experiments to evaluate our proposed methods for live adaptation of PETs. In this section, we will introduce the experiment settings and methods. In the following evaluation runs, we do not test the pipeline with a demonstration case similar to the real-world scenarios. There are two main reasons. Firstly, in the real-world scenarios, situational change is not frequent. Less data can be obtained from a single trial with real-world scenarios. On the other hand, the underlying stream processing platform, Apache Flink, operates in a multithreaded manner, where the exact execution timing is non-deterministic. In this case, only through larger amount of measurement can the statistical properties of out framework be obtained. Secondly, as is given in Table 5.2, we plan these experiments so that we can focus on the impact brought by a certain design aspect. The real-world scenarios contains multiple data sections, resulting in more parallel executing privacy protection tasks. The increased load of privacy protection computation could affect the experiment negatively. Take the first row in Table 5.2 as an example, if the impact of concurrent PETs of the same data section is under investigation, the number of the PETs is controlled. If there exists other computation task for other data sections, they might compete with the task under experiment for computational resource. Due to these two reasons, we elaborated special testing scenarios, so that we can focus on the experiment purpose and gain as much result as possible.

Our experiments are conducted with the Apple M1 Pro processor with 8 cores. The operating system is macOS 14.0. The Java library for the implementation is JDK 15. The version of Apache Flink is v1.17.0, Apache Kafka 3.5.0 and MongoDB 4.10.2.

5.3.1 Adaptation Comparison: Output Filtering and Dynamic PET Loading

In this section, we present the experiment result of the first row in Table 5.2, where we compare the adaptation approaches of output filtering (Variant 1 from Figure 3.6a) and dynamic PET loading (Variant 3 from Figure 3.6c). Through this experiment, the impact of the total number of concurrently running PETs on the data latency, as well as the impact of switching steps of Variant 3, as depicted in Figure 3.7, on the switching decision latency are revealed.



(a) Unprocessed Image

(b) Processed with “pixelate” mode

(c) Processed with “blur” mode

(d) Processed with “blacken” mode

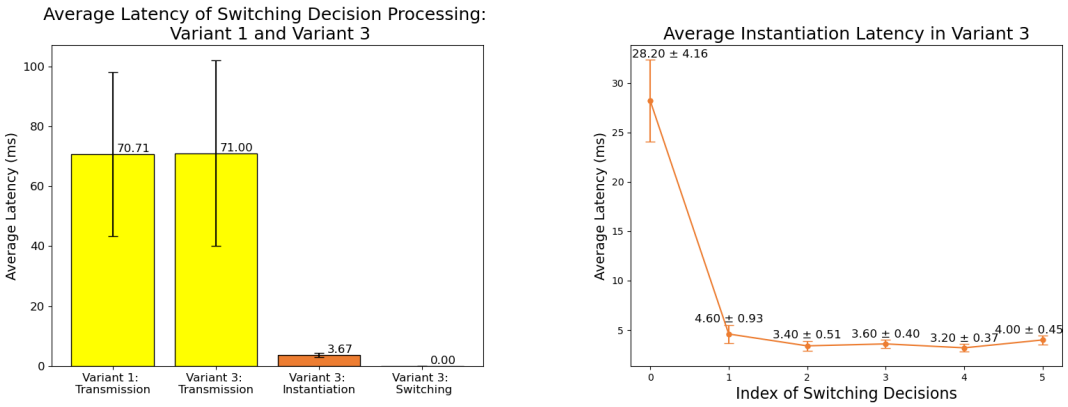
Figure 5.4: Different Processing Mode of the Applied PET for Image Data [Hel22]

In this set of experiments, the pipeline of Variant 1 will be constructed with different number of participating PETs. On the other hand, Variant 3 will switch between the same set of PETs. In order to highlight the experimental effect, the image data are selected as the target data section since the images are featured with higher refresh frequency and larger data volume. An excerpt of the raw image data from Geiger et al. [GLSU13] are used in this set of experiments.

The experiment setup is as follows. The input vehicle data are images, which are from an excerpt of the raw image data from Geiger et al. [GLSU13]. Each image has a resolution of 1226×370. The refresh rate is set to 10Hz, which corresponds to the shooting frequency of this set of picture sequences. The experiment will run for 30 seconds. As for the PETs, we apply those for image data proposed by Held [Hel22]. This PET algorithm is able to make sensitive information in an image unrecognizable. It processes input image by using image segmentation as the first step. Then the detected objects are categorized into classes with different priorities. A threshold value is given to determine which classes of objects to be made unrecognizable. In Figure 5.4 some example of the processing result is shown. The PET provides three processing modes: pixelate (Figure 5.4b), blur (Figure 5.4c) and blacken (Figure 5.4d). In this experiment, PET with different parameter combination are considered as individual PET algorithms. They will be switched against each other. We used three parameter combinations, each represent one processing mode from Figure 5.4b to Figure 5.4d. As is described in the introduction of Section 5.3, we would like to obtain as much measurement as possible from one trial. In order to trigger the adaptation more frequently, the triggering condition is set to every 50 frames. This is realized programmatically. Every 50 frames, a switching decision is generated, containing the annotation of a different PET algorithm.

In the following, we will analyze the gained results from multiple measurement sessions. First, we look at the impact of the switching steps from Figure 3.7 on the switching decision latency. Then, we analyze the impact of the concurrent candidate PETs on the data latency.

The results of switching decision latency is shown in Figure 5.5. The results are derived from experiments where 3 PETs are switched against each other. Figure 5.5a depicts the latency components in both variants in bar plot with standard deviation. Recall the quantitative analysis of the processing of switching decision given in Equation (5.1) and Equation (5.3), the switching decisions in both cases experience one time transmission between the operators. In Variant 1, it is

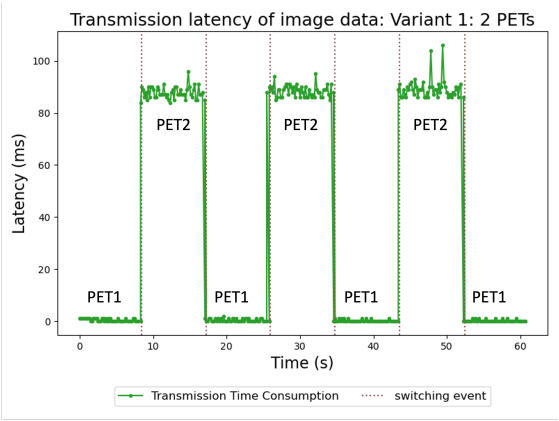


(a) Statistical Latency Value of Each Component **(b)** Chronological Statistical Latency of Instantiation

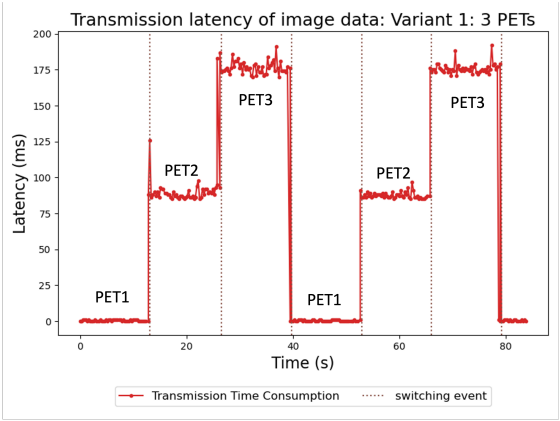
Figure 5.5: Statistical Latency Value of Components in Switching Decision Processing: Variant 1 and Variant 3

transmitted from the `Situation Evaluator` to the `Selector`. In Variant 3, it is transmitted from the `Situation Evaluator` to the `PET Enforcement`. The first bar on the left is the transmission latency from Variant 1, which is the only component of the switching decision latency in this case (ref. Equation (5.1)). The second bar on the left is the transmission latency from Variant 3. It's clear to see that the averaged transmission times in Variant 1 and Variant 3 lie closely to each other, which stands in alignment with our theoretical analysis. The concrete value of each switching decision fluctuates strongly, as indicated by the error bars in Figure 5.5a. This phenomenon is attributed to the scheduling and RAM access of the CPU cores. The uniqueness of Variant 3 is the additional effort of instantiation and switching. The switching latency of Variant 3 is shown in the rightmost bar, which is below 1 ms and therefore appeared as 0 ms. The second bar from the right indicates the instantiation latency. In this experiment settings, the extra steps in the switching decision processing are the instantiation and switching step. These two factors increased the latency of switching decision of Variant 3 by about 5%, compared with the latency in Variant 1. It is worth noticing that the instantiation latency shows the behavior of converging to a steady state during the operation. The measured instantiation latencies in multiple trials are plotted in Figure 5.5b. Since the simulation time is 30 seconds and every 5 second a switching decision is issued, there are 6 switching decisions in each run, which are indexed from 0 to 5. The mean value and standard deviation of the measurement belonging to the same switching decision are calculated and plotted against the index. As is shown in Figure 5.5b, the instantiation latency of the first switching decision is the spike of latency curve. We have conducted multiple experiments, where the first switching decision comes right after cold start of *AdaPrivFlow* or after certain time of steady operation, this spike remains existing. The possible reason for this phenomenon lies in the internal operating mechanism of Flink. In the statistics shown in Figure 5.5a, this data point is considered as outlier and therefore excluded from the statistical results.

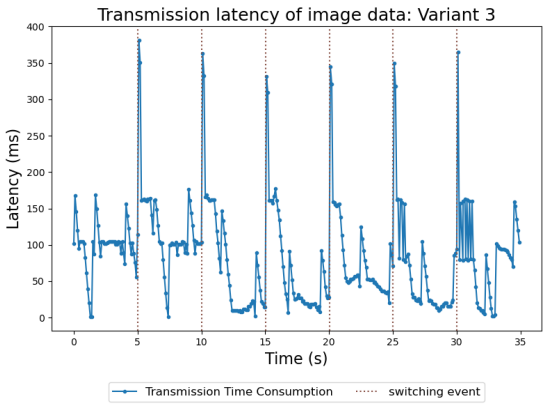
Up to this point, it could be concluded that the impact of switching steps of Variant 3, as depicted in Figure 3.7, is subtle. Only an increased latency of 5% is observed, compared with Variant 1.



(a) Variant 1 with 2 PETs



(b) Variant 1 with 3 PETs



(c) Variant 3

Figure 5.6: Transmission Latency and Evaluation Latency in Variant 1 and Variant 3

In the following, the impact of concurrent candidate PETs on data latency will be presented. Recall Equation (5.4), three latency sources determine a *trigger*'s end-to-end latency: (a) T_{TR} transmission from situation evaluation to PET enforcement operator, (b) T_{PR} processing by the PET, and (c) T_{EV} evaluation time. Here we will present our key findings out of the measurement of these timestamps.

To investigate the impact of the number of concurrent candidate PETs, we built the two pipeline of Variant 1, each with 2 PETs and 3 PETs. Correspondingly, experiments with Variant 3 are conducted with 2 and 3 PETs. The results are shown in Figure 5.6. Figure 5.6a and Figure 5.6b shows the transmission latency from Variant 1 with two and three PETs respectively. Figure 5.6c shows the curve from a general configuration of PETs from Variant 3, since the experiment result shows that the number of the concurrent PETs from the same data section makes barely impact, which is in alignment with our theoretical analysis in Section 5.2. As is shown in Figure 5.6a and Figure 5.6b, with the increase of the number of concurrent PETs, the maximal transmission latency among all the concurrent PETs increases in a staircase-manner. we can observe the following pattern: if there are n concurrently running PETs, the transmission time of data records from

situation evaluator (SE in Figure 5.1) to each PET enforcement operator (PE in Figure 5.1) can be expressed as $\mathsf{Trans}\{SE, PE\} = k(i-1), i = 1, ..., n$, where k is a constant. A details analysis of this phenomenon is presented in Section 5.4.1.

The behavior of transmission latency in Variant 3 is shown in Figure 5.6c. The measured value of the vehicle data is featured with spikes at the *triggers*. This can be explained through the operations to ensure the accurate PET enforcement, as is proposed in requirement R3 and Section 3.4. Due to the extra computational steps and potential buffering of the aggregated vehicle data, a spike is likely to occur. The overage latency across all aggregated vehicle data lies in the range of 90 ms to 100ms. The latency in this range also appeared in Figure 5.6a and Figure 5.6b. In both figures, the transmission of when `PET2` is applied. The fluctuation, which is not appeared in Figure 5.6a and Figure 5.6b, might be resulted from the internal stream control mechanism of Apache Flink.

A very apparent issue in Figure 5.6a and Figure 5.6b is that the switching event have shifted from the time when they should have occurred, while they take place every 5 seconds punctually in Figure 5.6c. Hence, the true refresh rate of the data source should be observed, and the result is shown in Figure 5.7.

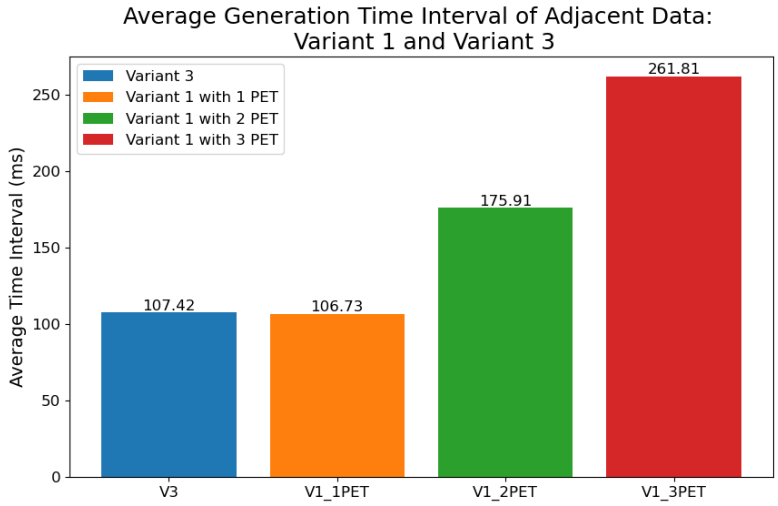
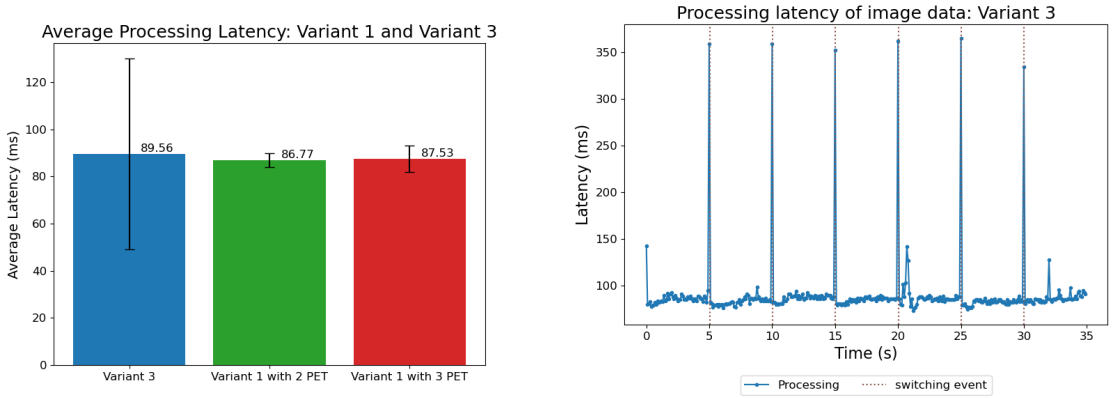


Figure 5.7: Impact of Number of Concurrently Running PETs on the Flink Sources

Figure 5.7 shows the true interval of data generation in the applied Flink sources. In this set of experiments, the image data has a refresh rate of 10 Hz. The theoretical interval of the adjacent image data should be 100 ms. In this figure, beside the pipelines under investigation, we also conducted the experiment on Variant 1 with 1 PET. This pipeline serves as the control group, since Variant 1 with 1 PET has essentially the same topological segment as Variant 3 between SE and PE, as is depicted in Figure 5.1 and Figure 5.2. As is depicted in Figure 5.7, the result from Variant 3 and Variant 1 with only 1 PET can approximately match the theoretical value. However, Variant 1 with only 1 PET is not meaningful, since nothing would be switched in this case. In contrast, in the meaningful applications of Variant 1, where the pipeline is integrated with multiple candidate PETs, the generation frequency cannot stick to the defined refresh rate. If these pipelines are utilized in real-world application, due to this phenomenon, some frames must be removed through preprocessing to reduce the refresh rate at the ingestion. This might lead to information

lost. Based on the experiment result on our testing environment, it can be concluded that Variant 1 is not suitable for real-world application. More analyzation about the reduced ingestion rate is given in Section 5.4.1.

After analyzing the transmission latency, the processing latency is addressed. Figure 5.8a shows the processing latency of each involved pipeline. It's clear to see that the number of concurrently running PETs has barely impact on the processing latency. We limit this conclusion to the multicore computing system, where each PET is executed on a distinct core. The standard deviation of the measurement from Variant 3 is larger than the rest variants. It can be explained that the steps of PET instantiation and switching have negative impact on the statistics, since during the transition the throughput of the PET enforcement operator is slowed down, which disturbed the steady operation of the whole pipeline. Thus, the processing of these triggering data are affected. The detailed measurement is depicted in Figure 5.8b, where the value of processing latency is plot against each occurring switching decision. As is shown in the figure, every time a switching decision is issued,



(a) Impact of Concurrent Running PETs on the Processing Latency

(b) Temporal Progression of Processing Latency in Variant 3

Figure 5.8: Statistical Latency Value of Each Component by Switching Decision Processing in Variant 1 and Variant 3

the processing latency reaches a local maxima. Shortly after the transition, the processing rate returns to its normal level.

The following points can be concluded from this set of experiments:

- The number of concurrently running PETs has huge impact on the overall performance of the pipeline. Two running PETs for image processing already prevent the pipeline from maintaining the proper 10 Hz rate of data ingestion from the source. Therefore, Variant 1 is not suitable to be utilized as the pipeline for *AdaPrivFlow*.
- The instantiation and switching latency in switching decision latency in case of dynamic PET loading contributes limited increase of the processing time of switching decisions. Therefore, the impact on the data latency is also limited.

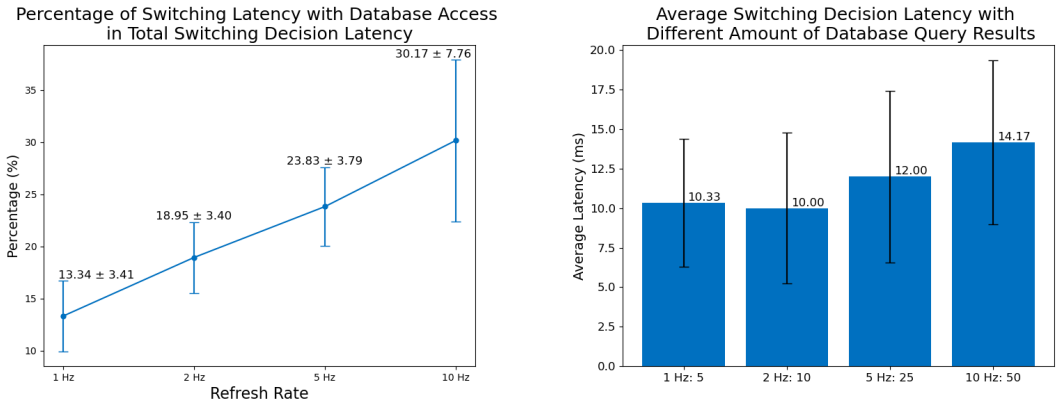
5.3.2 Influence of Database Access

In Section 5.3.1, the impact of the instantiation step and switching step by dynamic PET loading is investigated. As depicted in Figure 5.2, the introduced additional latencies also include the latency due to database access for state building of stateful PETs, besides the two latency sources mentioned previously. In this section, we perform a set of supplementary experiments to investigate the impact brought by database access. The number of necessary data records to build the state is varied, so that the impact of data volume can be observed.

The experiment setup is as follows. To elaborate a straightforward testing scenario, we use the single input data section speed to perform the experiments, since stateful operations, such as aggregation, can be easily performed on scalar data. The speed value is set to be a sine function with a period of 10 seconds, beginning from 0 km/h and varying between 0 km/h and 60 km/h. It can be expressed as $v(t) = 30\sin(\frac{\pi}{10}t - \frac{\pi}{2}) + 30, t \geqslant 0$, where the speed is denoted as v, and t is the elapsed time from experiment start. As for the PETs in use, we use a *dummy* PET for real PETs. A dummy PET is a class that implements the PET interface in Section 4.3.1. It also has an annotation in the illustrated format in Section 4.3.2. By doing so, the latency of the switching decision can be evaluated the same way as a real PET algorithm. In this setup, two dummy PETs are defined. The stateful one computes the averaged value, relying on the history data from the past 5 seconds. The stateless one simulates the deactivation of privacy protection due to safety reasons. It doesn't process the input vehicle data. Under the speed of 30 km/h, the stateless dummy is applied. If the speed equals or greater than 30 km/h, the stateful dummy is applied. With these settings, totally ten switching procedure are to be triggered per run, five of which reveal the behavior of switching to stateful algorithms. According to multiple online resources about the technical specification of speed display on the market, the refresh rate of speed reading lies in the range between 0.5 Hz to 10 Hz. Therefore, we vary the refresh rate of the speed data by 1 Hz, 2 Hz, 5 Hz and 10 Hz. These correspond to 5, 10, 25, and 50 history data records for state building, respectively.

In the analysis of the experimental result of this part, we drop the results of Variant 1. We already drew the conclusion at the end of Section 5.3.1 that Variant 1 is not suitable, since it is very sensitive to computational load and can therefore easily fail to ingest data at the target refresh rate. We further ignore the investigation of the ingestion rate of Variant 3 for this experiment, since its capability of holding the target ingestion rate is already proven with the image data, which is of higher dimension and has longer processing time. Only the switching latencies containing database access with different amount of retrieved data stay in focus.

Figure 5.9 illustrates the impact of amount of history data on the switching decision latency. Figure 5.9a illustrates the percentage of state building latency in the switching decision latency with different amount of retrieved history data. It's clear to see that with increased amount of history data, the state building latency contributes more to the switching decision latency. The result is plausible, because a higher time demand for querying larger amount of data in the database is expected. Recall the result in Figure 5.5a, the switching latency without database access in case of the stateless PET is less than 1ms. In Figure 5.9b, the averaged latency due to database access under different amount of retrieved data is shown. It can be seen that the time of database access is limited below 15 ms by 50 history speed records. Considering that fact that the retrieved data are scalar values and hence of very small size, the time consumption of retrieving other types of data with larger size will increase. An experiment with image data is not included, since stateful (dummy) PET for images is not intuitive. Nevertheless, it is very cost-effective to trade space for



(a) Percentage of State Building Latency in Switching Decision Latency

(b) Average Switching Latency with Different Amount of Database Query Results

Figure 5.9: Latency of Switching Step with Database Access

time through an extra step of database querying, since a stateful PET doesn't need to be cold started wait for the incoming records to warm itself up. A cold start could even be unfeasible, if the history data has large volume while being refreshed infrequently.

As a summary, the integrated database and its access for the provisioning of stateful PETs is very effective. Our approach can shorten the down-time of the system as much as possible.

5.3.3 Capability of Handling Data with Higher Refresh Rate

In the experiment results in Section 5.3.1, the measured processing latency stabilizes at about 90 ms, which is not a severe problem for a refresh rate of 10 Hz. However, as Geiger et al. [GLSU13] pointed out, they record the images with a refresh rate from 10 Hz to 100Hz. The centralized PET enforcement is not able to handle higher refresh rate than 10 Hz. The capability of *AdaPrivFlow* of handling data with large volume and high refresh rate should be explored. In this section, we will conduct a set of experiments to evaluate the effectiveness of *Stateless Distributed PET Enforcement* as an optimization.

The experiment setup is as the following. Image data is the appropriate data section to fulfill our experiment purpose. Same as Section 5.3.1, the data originates from Geiger et al. [GLSU13]. Per trial 1100 frames flow through the pipeline. The refresh rate varies from 10 Hz to 100 Hz. The same set of PETs in the experiments of Section 5.3.1 is used. For simplicity, two PET algorithms from the set are switched against each other. The switching decisions are issued after *AdaPrivFlow* operates stably for a period of time. Recall the result in Figure 5.6c, the operation of the pipeline will be disturbed because of the switching decision. Consider the large amount of data flow to be tested in this set of experiments, the disturbance would be enlarged. A non-steady pipeline could produce unreliable results. On the other hand, situational change in real-world scenario is infrequent. It is reasonable to assume that every switching event takes place in a steady operation state of the pipeline. With these arguments, the switching condition is set to be the steady operation after a period of time.

Since the purpose of this experiment is to evaluate the effectiveness of distributed PET enforcement, the optimal utilization of computational resources should be taken into account. While a stream of image data feasible with 2 parallel instances will also work with 4, the excessive allocation leads to waste of computational resources. In a real-world application scenario, there are not only image data being protected through the framework. By minimizing the number of parallel instances, the saved computational capacity can be utilized by parallel processing of other data sections. Consequently, we will evaluate the performance with the following criteria: for a given refresh rate, the lowest number of necessary parallel instances to perform distributed PET processing while maintaining the target ingestion rate.

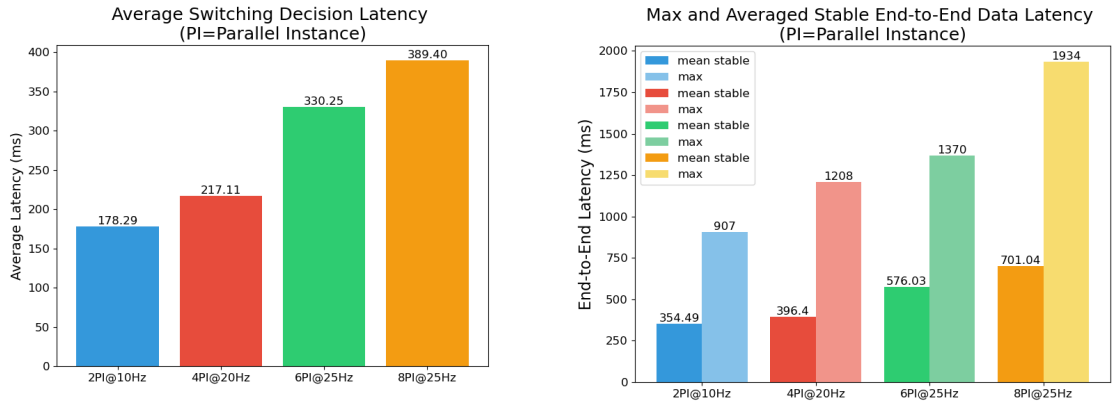
Table 5.3: Lowest Computational Resource Consumption by Different Image Data Refresh Rate

Index	Refresh Rate (Hz)	Number of Parallel Instances	Achieved Ingestion Rate (Hz)
1	10	1	9.3 (ref. Figure 5.7)
2	10	2	10
3	20	4	20
4	25	6	24.75
5	25	8	24.38
6	30	8	25.6

Table 5.3 shows the result under our testing environment mentioned in the introduction of Section 5.3. The result in the row of Index 1 is derived from Section 5.3.1, where the image data stream with 10 Hz refresh rate was processed with a single instance of `PET Enforcement` operator. It serves as a reference for the results in other rows. It can be observed that highest achievable refresh rate of the applied image data is 25Hz (Index 4). In this run, 6 parallel instances for PET enforcement are integrated into the pipeline. In the row of Index 6, where the refresh rate reaches 30 Hz, even if each core is allocated with a parallel instance, the ingestion rate cannot be maintained at the target rate, hence this row listed is colored with red background. Due to this reason, the need of testing frequencies higher than 30 Hz could be saved. However, this limit is dependent on the computational power of the device. The capability of handling 30 Hz image data could be realized by utilizing CPU with more cores and higher frequency.

In the row of Index 5, the result of 8 parallel instances at 25 Hz frequency is also listed with the configuration of 6 parallel instances at 25 Hz (Index 4). In this case, the ingestion rate decreases, even though more parallel instances is integrated into the pipeline. This can be explained by the full load of computational intensive tasks on the CPU cores. since the number of the parallel instance equals the number of CPU cores in our experiment environment,

Figure 5.10 shows the measured latency of switching decision (Figure 5.10a) and data (Figure 5.9b) with the configurations of the lowest computational resource consumption listed in Table 5.3. These figures present the result of Index 2 to Index 5. In Figure 5.10a, with the increase of the number of parallel instances, the latency of switching decision, impacted by the additional synchronization phase in distributed PET enforcement, shows a rising trend. This phenomenon is plausible, because of the increased number of feedback signal transmission. In Figure 5.9b, the data latency is represented by the mean latency during steady operation and the maximal latency during the run. Consider the result in Figure 5.6c, the *trigger* and a few data afterwards is affected by the switching procedure, forming a spike in the diagram. In distributed PET enforcement, the *trigger* undergoes



(a) Switching Latency of Distributed PET Enforcement with Different Number of Parallel Instances

(b) Max and Averaged Stable End-to-End Data Latency in Distributed PET Enforcement

Figure 5.10: Data and Switching Decision Latencies in Configuration of the Lowest Computational Resource Consumption

buffering in the `Controlled Buffer` resulting in a spike in the temporal progress of latency, as is described in Section 3.4.3. Due to this fact, the mean latency is calculated with exclusion of the latencies from the affected data. The maximal latency is taken from the highest spike. With the increase of flow volume, the data latency demonstrates an increasing trend.

The conclusion of this set of experiments should consider the overall performance of the pipeline. It could be observed clearly that the latencies of switching decision and data is one order of magnitude higher than the generation period of the images. For example, in the case of 4 parallel instances under 20 Hz refresh rate, the images are generated every 50ms, while the latencies of switching decision and data are 4 times and 6 times higher, respectively. Nevertheless, the pipeline is still able to process the data on-time. As is shown in Table 5.3, the ingest rate remains 20 Hz, corresponding to the target refresh rate.

As a summary, regarding the capability of handling high dimensional data with high refresh rate, our proposed pipeline for the distributed PET enforcement has the limit of images (1226×370) at 25 Hz.

5.3.4 Conclusion of Experiments

Across all the tested variants and configurations, the component with the largest proportion is the duration of transmission from one operator to another. This includes not only the measured transmission from the situation evaluator (SE) to the target operator, i.e., PET enforcement (PE) or Selector (SL) depending on the pipeline, but also the synchronization phase of switching decision and the duration for presentation by distributed PET enforcement. The latency due to instantiation of PET algorithm occupies a slight proportion. The switching latency of stateless PETs, where the access to local persistent storage is omitted, can be ignored. Although the storage access contributes to the latency in the case of stateful PETs, the time required as far less than waiting for the same amount of data flowing into the pipeline. This greatly reduces the “down-time” of the framework.

Although the processing of data records and switching decision is impacted greatly by the transmission time, *AdaPrivFlow* still excels the solution of Variant 1 from Figure 3.6a, where all the candidate PETs are integrated. Our framework excels not only in dealing with dynamic privacy policies, but also in its resilience against the total number of candidate PETs, the dimension, and the refresh rate of the vehicle data. With access to the data store for history data, the adaption duration is increased by less than 15ms while saving the waiting time for 50 scalar data records. Our framework can achieve the privacy protection on image data updated every 25Hz by using distributed PET enforcement. Furthermore, the data sequence in the output is retained.

5.4 Discussion

In this section, we discuss the experiment result from the experimental data in Section 5.3. Firstly, we summarize the learned lessons from the result. Then we discuss the limitation of our approach.

5.4.1 Lessons Learned

Through an in-depth observation of the experiment result in Section 5.3, we are able to peek into some details about the principle of Apache Flink. The conclusions are summarized as learned lessons, so that they could help future research in the direction of performance optimization.

The first observed operation principle of Apache Flink is operator chaining. In Figure 5.6 we have observed a staircase-like progression of transmission latency in the pipeline of Variant 1. It can be observed that the transmission time of data records from situation evaluator (SE in Figure 5.1) to each PET enforcement operator (PE in Figure 5.1) can be expressed as $\mathsf{Trans}\{SE, PE\} = k(i-1), i = 1, ..., n$, where k is a constant. Recall the pipeline topology presented in Figure 5.1, multiple PET operators are connected to the situation evaluator. This topology requires the data emitted by the upstream operator to be copied multiple times. The value of k probably represents the necessary time for a data record to be copied and emitted by an operator into the memory and fetched by the proceeding operator. The summarized rule reflects the probable compiled topology internal of Flink. That is, one PET enforcement operator is chained with its upstream neighbor, which means, instead of being transmitted via the memory, the data is directly handed over to the next downstream operator. This is the strategy provided by Flink to optimize the performance [The23c]. Other PET enforcement operators are not chained and gain the input from their upstream via memory access.

The second observed operation principle is Flink's handle of *backpressure*. When the computational load increases, the phenomen *backpressure* can be observed. The green and red bar in Figure 5.7 have demonstrated that the data generation in the pipeline of Variant 1 with multiple PET is unable to maintain the target rate at 10Hz. This phenomenon can be attributed to backpressure. According to Ververica [Ver15], under the condition of backpressure, the source is throttled to adjust to the slowest component of the pipeline, letting the pipeline reach a steady state. In our experiment, the `Source Function` is throttled. This results in the green and red bar with decreased data generation rate in Figure 5.7

The third learned lesson is to leave a margin of computational resource to the CPU. In the row of Index 5 in Table 5.3, where 8 parallel instances are applied, the ingestion rate is decreased compared with 6 parallel instances under the same image refresh rate of 25 Hz. This can be attributed to the

CPU cores being fully occupied by computationally intensive tasks, since the number of parallel instances matches the number of CPU cores in our experimental environment. The background tasks of Flink compete with the streaming job for computational resources, which consequently slows down the pipeline. This observation suggests an important takeaway: it's not advisable to fully occupy the CPU in real-world applications, because there would be PETs running for various data sections. If the PET enforcement of a specific data section requires parallelism equal to the number of CPU cores, it would inevitably lead to competition for computational resources with other data sections. This competition can result in the deceleration of the operator, and due to backpressure, the entire framework.

5.4.2 Limitation

The limitations of *AdaPrivFlow* can be discussed in two aspects: the limitation of the design and of the implementation.

Design Limitation

In total, the limitation in design can be concluded in two points.

The first point is the consumption of computational resource due to design, which can be concluded without conducting any experiments. There are opportunities to optimize the utilization of computational and memory resources. *AdaPrivFlow* employs a static pipeline for parallel privacy preserving across multiple applications. As is depicted in Figure 4.1, the data records are fanned out for (a) pipelines running in parallel for each application, and (b) parallel evaluations for distinct data sections within a single application. In the stream processing jobs, the increased volume of data are transmitted via network, which escalates the demands on memory and bandwidth. While the framework's design minimizes computational resource wastage by allowing only relevant PET algorithms in computations, this waste of computational resources is not entirely eliminated. Since the framework arranges an independent pipeline for each involved application, there is no exchange of data between them. In the applicable policies for several applications, scenarios arise where subsets of applications employ identical PET algorithms under the same conditions. Consider the following example. Suppose the navigation app and the fleet management app are both set to utilize a certain obfuscation algorithm A when the car is inside the city. In *AdaPrivFlow*, the PET algorithm A is independently activated in the pipeline of both apps. This results in redundant computation of the identical data records. To fully eliminate the waste of computation resource, a dynamic pipeline should be implemented. This would allow different applications to share results from commonly utilized PET algorithms and depending on the situation, transition to app-specific defined PETs.

The second limitation lies in the `Key Assigner` in the pipeline segment of distributed PET enforcement (ref. Figure 3.14 and Figure 5.3) The limitation is observed through the results in Table 5.3. The rows from Index 3 to Index 4 illustrated that a linear increase of number of parallel instances is not sufficient to deal with the extra computational task brought by a linear increase of the data refresh rate. From 20Hz to 25Hz, the necessary number of parallel instances increased 50%, while the data stream volume rose only 25%. Recall the phenomenon *backpressure* mentioned in Section 5.3.1, the potential reason is the bottleneck operator in the pipeline. In order to investigate the position of

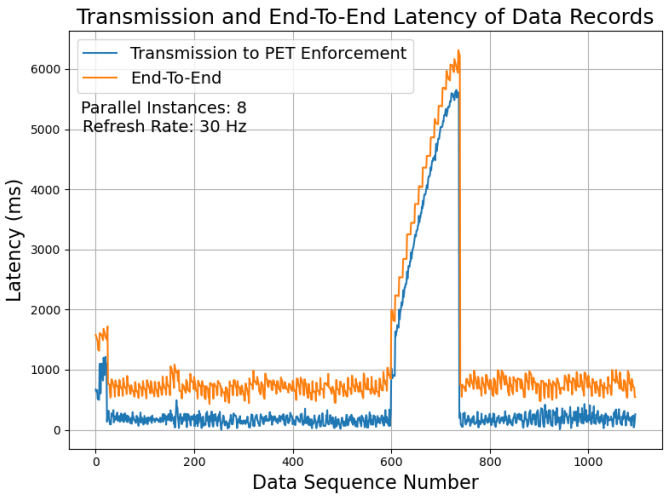


Figure 5.11: Data Latency Components in a Streaming Job Failed to Keep the Ingestion Rate

the bottleneck in the pipeline topology, the plot of the latency components over the frame sequence number is shown in Figure 5.11. Here the configuration in row Index 6 is chosen to be presented to highlight the bottleneck effect, since this run fails to maintain the target ingestion frequency of 30Hz. In this plot, the transmission latency (blue) and end-to-end latency (orange) of each image frame are plotted against the frame sequence number. Recall the experiment settings for distributed PET enforcement, the switching decision is issued after a period time of the pipeline's steady operation. In this trial, the switching decision is set to be released at the 600th frame. As is shown in Figure 5.11, from frame number 600, a sudden rise of the shown latencies can be seen. This is plausible due to the issued switching decision. Before the *trigger* and after the system returns to steady operation state, the vertical distance between the two lines of latency measure at each sampling point has not changed significantly Recall the depicted latency measurement points in Figure 5.3, the transmission latency is given by $t_{p1} - t_d$, while the end-to-end latency is represented as $t_e - t_g$. The difference between the two value is the summation of other latency sources. In this case, there are no significant changes of them. As is already illustrated in Figure 5.5a, the switching latency for stateless PET is below 1 ms, which is also the case for the experiments in this section. Therefore, the transmission latency measures the duration from SE to PE, as is shown in Figure 5.3. Obviously, the data flowing into the distributed PET enforcement pipeline undergo the additional KA operator, which assigns a *key* to each data for data order restoration, as is described in Section 3.4.3. The operator KA is the bottleneck, which is overloaded with the release of buffered data records during the processing of switching decisions. A better approach of order restoration in the distributed PET enforcement would be needed to overcome the bottleneck effect shown in KA operator.

Implementation Limitation

The limitation in the aspect of implementation lies in the restriction due to the underlying stream processing platform. As is already pointed out in Section 5.3.4, the major latency source is the transmission between the operators. However, as *AdaPrivFlow* proves its feasibility based on the established platform Apache Flink, many underlying technical details cannot be arbitrarily

controlled. If we want to delve into the underlying principles and perform in-depth optimization for our scenarios, the time consumption would be enormous, and this is not within the scope of this work as the first step of proving its feasibility. But the optimization of performance could be a promising evolving direction in the future researches.

6 Related Work

In this section, a few literatures that are technically related to our work are presented. Our study has two aspects: privacy protection through PETs in CVs, and adaptive algorithm switching in stream processing systems. The relations of other works to ours are based on these aspects.

Various works cover the area of privacy protection in CVs from different perspectives. Not every study of privacy protection in CV takes adaptivity into consideration. There are existing studies protect the privacy through other means instead of applying PETs. For example, Dietzel et al. [DKSK12] protects the privacy in intelligent transportation systems by designating a sandbox, where the privacy-relevant properties of the application inside the sandbox could be analyzed and enforced. Herges et al. [HAK+12] protects the vehicle data by elaboration of an access control framework. Their work is implemented based on Android and applied to telematic applications. Although this framework utilized other mechanism to protect the privacy, it introduces minimal driver distraction, which corresponds to part of our requirement in R1. However, unlike our study, these studies didn't take adaptivity into consideration. Some studies concentrate on a certain aspect of the data protection in CVs. For example, Lu et al. [LLG12] dealt with privacy preservation in the perspective of authentication in vehicular ad-hoc network with adaptive pseudonyms. Unlike our study, this work involves communication with external systems. In a broader domain of Internet of things (IoT), various existing studies concentrate on a certain data type. For instance, Elkhodr et al. [ESC13] focused on the Location-Based Services (LBS) in IoT with a context-aware adaptive approach. On the contrary, our study supports the common data types in a CV, providing a comprehensive privacy protection. Another related work in the IoT is the research of Schaub [Sch14]. He investigated dynamic privacy adaptation in ubiquitous computing by leveraging contextual awareness. His research is featured with the regulation of personal privacy. However, rather than being rule-based, the decision is made with respect to the user's privacy preference. Moreover, the core of the adaptation to determine the user's likely privacy preference through privacy reasoning, which deviates from the research focus of our work.

In the following, we shift the attention to the related works applying stream processing technology for the aim of privacy protection. Wang et al. [WRW+22] has contributed in the IoT domain. In their work, a framework for IoT streaming data is proposed, which can prevent sensitive inferences and preserve data utility. In order to realize real-time transmission and fast processing, this framework is deployed on a distributed streaming platform. However, their work doesn't incorporate adaptation of the privacy protection methods during the operation.

Qin and Eichelberger [QE16] focused on algorithm switching for adaptive stream processing systems, which stands technically closely with our research focus. Her work presented a quality specification for the algorithm switching and explored various switching approaches, taking into account switching time, switching gap and throughput disturbance duration. The concept was implemented using Apache Storm. Her research proves the feasibility of our work. Moreover, the quality assessment in her research has greatly inspired the definition of the requirements R3

and R5 in our research. However, it mainly described a general-purpose algorithm switching framework. Some specifications in the context of privacy preserving in modern CVs are not addressed. Furthermore, this framework differs from our work in that the cause for the switch is not covered. It emphasizes on the switching mechanism.

Apart from the work with general purpose, such as Qin and Eichelberger [QE16], there exist frameworks in the logistics domain that switch a fragment of their processing logic with context-awareness. Bucchiarone et al. [BMPR12] proposed a context-aware framework for dynamic adaptation of business processes to deal with real-world situations. This framework is distinguished by the utilization of process fragments, which models the reusable process knowledge. In relation to our work, the modularized PETs can be equated to reusable process knowledge.

As of this writing, we haven't found studies that deal with our specific challenge via a different approach. Therefore, our work differs from the other previous related works in that we provide a framework that is able to provide comprehensive and adaptive privacy protection using PETs based on the change of external environment and own state.

7 Conclusion and Outlook

Privacy protection in the context of CVs has been a popular research area. As review in Chapter 6, various literatures in this area can be found. However, they are limited to protection of a certain aspect of the whole spectrum of the data types in a CV. Moreover, these researches didn't consider the dynamic nature of privacy protection. They focus on the privacy protection technologies under certain circumstances. In the reality, the demand of privacy protection is situational, as is indicated by the typical scenarios in Chapter 1. To realize a comprehensive privacy protection of vehicle data, while meeting the varying needs under the changing situations, we proposed *AdaPrivFlow*, a privacy-enabling framework utilizing stream processing technologies.

In this work, we proposed the requirements for such a framework. Based on the requirements, we illustrated the architectural concept of *AdaPrivFlow*. In the architecture, the data pipeline consists of 4 components: the `Input Sources`, the `Situation Evaluator`, the `PET Enforcement` and the `Data Sink`. Beside the data pipeline, a database is integrated, considering the fact that *stateful* PET algorithm rely on history records to process the vehicle data. The potential adaptation approaches suited in this architecture are explored and finalized as dynamic PET loading, where the `PET Enforcement` components of the pipeline load different PETs based on the instruction from the `Situation Evaluation` component. This adaptation choice requires the `PET Enforcement` operator to be able to not only parse the information of the PET, but also instantiate and switch the PET at the right time. We defined the standards regarding the PETs. An interface for the PETs is designed, so that the PETs can be easily changed through a modularized design. The format of PET annotation is defined, so that the information of the PET can be correctly parsed as the preparation of PET switching. The definition of the annotation and interface take the characteristics of the stateful PETs into account, so that these PETs can be correctly initialized with history data. In order to ensure that each vehicle data is enforced by the PET it complies to, we proposed the feasible condition of switching, which determines the correct time point of activating the newly initialized PET. The feasible condition is evaluated against the processing state of the switching decision and the appearance of its affiliating *trigger* data. The feasible condition is met only if the switching decision is processed and its affiliating trigger has appeared in the `PET Enforcement` operator. The operation based on feasible switching condition incorporates the *trigger marker* and *buffering* functionality. The proposed switching condition is up to this point applicable in one single `PET Enforcement` operator, which is called stateless/stateful centralized PET enforcement. In order to support data type with high dimension and high refresh rate, such as camera images, we proposed distributed PET enforcement. In this form, multiple parallel instances of the `PET Enforcement` operator are spawned with the same loaded PET, processing the input of the same data section. The feasible condition of switching is extended to achieve synchronized switching among the parallel instances. We also proposed the method of data order maintenance to keep the data ordered after being distributedly processed by multiple parallel instances. For proof-of-concept, we compared

and selected the potential underlying technologies to prove the feasibility of *AdaPrivFlow*. Apache Flink is selected as the underlying stream processing platform. MongoDB is selected to be the database.

AdaPrivFlow is able to react automatically to situational changes without restart or user intervention. At the same time, it supports individual dynamic privacy protection for each application. Its performance is evaluated against the adaption approach of output filtering, where all the candidate PETs participate in privacy protection and only the result of the applicable PET is kept. Our framework excels in three aspects. Firstly, our framework is capable of handling dynamic privacy policy with new PET. Secondly, it possesses robustness against the total number of candidate PETs. Thirdly, it can handle the diversity of the vehicle data types, especially the ones featuring by high data volume and high refresh rate. With the access to the database for history data, the adaptation for stateful PETs duration just increased less than 15ms, but saved the waiting time for 50 records of scalar values. With the application of distributed PET enforcement, *AdaPrivFlow* is able to protect the privacy in image data refreshed at 25Hz. In addition to this, the sequence of the image data remain preserved, which eases the follow-up query, such as video playback or live broadcast.

Outlook

Due to the time constraint of this master thesis, the conceptualization of *AdaPrivFlow* took place under several assumptions. These could be evolving directions for future work. Regarding the PETs, currently the supported PETs is limited to *ego-centric*, meaning that they do not need information exchange with other CVs. The PETs could be extended to the ones involving information exchange with other entities. What's more, the candidate PETs in the privacy policies are currently assumed to be available by the time it is about to be applied. The future work could enrich *AdaPrivFlow* through investigating the mechanism of PET provision from a trusted remote repository.

Regarding the framework itself, the component of current framework could be extended. As is pointed out in Section 4.1, the integration of in-memory cache database makes the database access more efficient. Due to time constrain and simplicity, this integration is not implemented.

Another potential future research direction of the framework structure is to start from the bottom of the stream processing platforms and dig deeper into their principles. Through the research on "lower level", on the one hand, the overall performance of our framework could be optimized. On the other hand, the feasibility and actual complexity of the dynamic pipeline (Figure 3.6b) can be explored. Since the current proof-of-concept relies on established stream processing platform, there are lots of restriction due to the enabling technology. Through low-level investigation, a more elaborated pipeline topology and optimized operation principle of the stream processing platform could be developed.

Furthermore, *AdaPrivFlow* could be integrated with a driving simulator as the first step of real-world application. Sensor readings could be derived from external systems, instead of being simulated with assumptions of perfection, such as being aggregated and chronologically in order.

Bibliography

[AC18] R. Al-Dhubhani, J. M. Cazalas. “An adaptive geo-indistinguishability mechanism for continuous LBS queries”. In: *Wireless Networks* 24.8 (Nov. 2018), pp. 3221–3239. ISSN: 1572-8196. DOI: 10.1007/s11276-017-1534-x. URL: https://doi.org/10.1007/s11276-017-1534-x (cit. on p. 23).

[BMPR12] A. Bucchiarone, A. Marconi, M. Pistore, H. Raik. “Dynamic Adaptation of Fragment-Based and Context-Aware Business Processes”. In: *2012 IEEE 19th International Conference on Web Services*. 2012, pp. 33–41. DOI: 10.1109/ICWS.2012.56 (cit. on p. 84).

[CBB+03] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, S. B. Zdonik. “Scalable Distributed Stream Processing.” In: *CIDR*. Vol. 3. Citeseer. 2003, pp. 257–268 (cit. on p. 21).

[CDN15] R. Cramer, I. B. Damgård, J. B. Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015. DOI: 10.1017/CBO9781107337756 (cit. on p. 22).

[Con23] Confluent. 2023. URL: https://docs.confluent.io/platform/current/streams/overview.html (cit. on p. 50).

[CTS14] L.-W. Chen, Y.-C. Tseng, K.-Z. Syue. “Surveillance on-the-road: Vehicular tracking and reporting by V2V communications”. In: *Computer Networks* 67 (2014), pp. 154–163. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2014.03.031. URL: https://www.sciencedirect.com/science/article/pii/S1389128614001480 (cit. on pp. 17, 40).

[DE13] F. K. Dankar, K. El Emam. “Practicing differential privacy in health care: A review.” In: *Trans. Data Priv.* 6.1 (2013), pp. 35–67 (cit. on p. 23).

[DKM+06] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, M. Naor. “Our Data, Ourselves: Privacy Via Distributed Noise Generation”. In: *Advances in Cryptology - EUROCRYPT 2006*. Ed. by S. Vaudenay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 486–503. ISBN: 978-3-540-34547-3 (cit. on p. 22).

[DKSK12] S. Dietzel, M. Kost, F. Schaub, F. Kargl. “Cane: A controlled application environment for privacy protection in its”. In: *2012 12th International Conference on ITS Telecommunications*. IEEE. 2012, pp. 71–76 (cit. on p. 83).

[DR14] C. Dwork, A. Roth. “The Algorithmic Foundations of Differential Privacy”. In: *Foundations and Trends® in Theoretical Computer Science* 9.3–4 (2014), pp. 211–407. ISSN: 1551-305X. DOI: 10.1561/0400000042. URL: http://dx.doi.org/10.1561/0400000042 (cit. on p. 22).

[DWW+23] F. Dettinger, H. Wei, M. Weiß, N. Jazdi, M. Weyrich. "Dateneffiziente Vervollständigung des Umgebungsmodells von autonomen vernetzten Systemen mittels Sensorfusion". In: June 2023, pp. 513–524. ISBN: 9783181024195. DOI: 10.51202/9783181024195-513 (cit. on p. 40).

[ESC13] M. Elkhodr, S. Shahrestani, H. Cheung. "A contextual-adaptive Location Disclosure Agent for general devices in the Internet of Things". In: *38th Annual IEEE Conference on Local Computer Networks - Workshops*. 2013, pp. 848–855. DOI: 10.1109/LCNW.2013.6758522 (cit. on p. 83).

[GG07] J. Gama, M. Gaber. *Learning from data streams. Processing techniques in sensor networks*. Jan. 2007. DOI: 10.1007/3-540-73679-4 (cit. on p. 20).

[GKMS01] A. Gilbert, Y. Kotidis, S. Muthukrishnan, M. Strauss. "Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries". In: (Sept. 2001) (cit. on p. 21).

[GLSU13] A. Geiger, P. Lenz, C. Stiller, R. Urtasun. "Vision meets Robotics: The KITTI Dataset". In: *International Journal of Robotics Research (IJRR)* (2013) (cit. on pp. 69, 75).

[GSH+21] G. M. Garrido, K. Schmidt, C. Harth-Kitzerow, J. Klepsch, A. Luckow, F. Matthes. "Exploring privacy-enhancing technologies in the automotive value chain". In: *2021 IEEE International Conference on Big Data (Big Data)*. 2021, pp. 1265–1272. DOI: 10.1109/BigData52589.2021.9671528 (cit. on pp. 22, 23).

[GSU+22] G. M. Garrido, J. Sedlmeir, Ö. Uludağ, I. S. Alaoui, A. Luckow, F. Matthes. *Revealing the Landscape of Privacy-Enhancing Technologies in the Context of Data Markets for the IoT: A Systematic Literature Review*. 2022. arXiv: 2107.11905 [cs.CR] (cit. on pp. 22, 23).

[HAK+12] D. Herges, N. Asaj, B. Könings, F. Schaub, M. Weber. "Ginger: An Access Control Framework for Telematics Applications". In: *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*. 2012, pp. 474–481. DOI: 10.1109/TrustCom.2012.169 (cit. on p. 83).

[HB95] R. Hes, J. Borking. "Privacy-Enhancing Technologies: The Path to Anonymity". In: (Jan. 1995) (cit. on pp. 17, 22).

[Hel22] D. Held. *Schutz der Privatsphäre in verteilten Software-Defined-Car-Anwendungen*. 2022 (cit. on p. 69).

[HMVV13] P. Hank, S. Müller, O. Vermesan, J. Van Den Keybus. "Automotive Ethernet: In-vehicle networking and smart mobility". In: *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2013, pp. 1735–1739. DOI: 10.7873/DATE.2013.349 (cit. on p. 20).

[HWS+17] P. Hirmer, M. Wieland, H. Schwarz, B. Mitschang, U. Breitenbücher, S. G. Sáez, F. Leymann. "Situation recognition and handling based on executing situation templates and situation-aware workflows". In: *Computing* 99.2 (Feb. 2017), pp. 163–181. ISSN: 1436-5057. DOI: 10.1007/s00607-016-0522-9. URL: https://doi.org/10.1007/s00607-016-0522-9 (cit. on p. 30).

[IAM+19] H. Isah, T. Abughofa, S. Mahfuz, D. Ajerla, F. Zulkernine, S. Khan. "A Survey of Distributed Data Stream Processing Frameworks". In: *IEEE Access* 7 (2019), pp. 154300–154316. DOI: 10.1109/ACCESS.2019.2946884 (cit. on p. 21).

[Jen23] M. R. u. Z. M. Jen Caltrider. *Es ist offiziell: Autos sind in puncto Datenschutz die übelste Produktkategorie, die wir je getestet haben*. 2023. URL: https://foundation.mozilla.org/de/privacynotincluded/articles/its-official-cars-are-the-worst-product-category-we-have-ever-reviewed-for-privacy/ (cit. on p. 17).

[KF16] S. Kamburugamuve, G. Fox. *Survey of Distributed Stream Processing*. Feb. 2016. DOI: 10.13140/RG.2.1.3856.2968 (cit. on p. 21).

[KLB20] N. Kaaniche, M. Laurent, S. Belguith. "Privacy enhancing technologies for solving the privacy-personalization paradox: Taxonomy and survey". In: *Journal of Network and Computer Applications* 171 (2020), p. 102807. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2020.102807. URL: https://www.sciencedirect.com/science/article/pii/S1084804520302794 (cit. on pp. 22, 23).

[LCZ+14] N. Lu, N. Cheng, N. Zhang, X. Shen, J. W. Mark. "Connected Vehicles: Solutions and Challenges". In: *IEEE Internet of Things Journal* 1.4 (2014), pp. 289–299. DOI: 10.1109/JIOT.2014.2327587 (cit. on p. 17).

[LHSM23] Y. Li, P. Hirmer, C. Stach, B. Mitschang. "Ensuring Situation-Aware Privacy for Connected Vehicles". In: *Proceedings of the 12th International Conference on the Internet of Things*. IoT '22. Delft, Netherlands: Association for Computing Machinery, 2023, pp. 135–138. ISBN: 9781450396653. DOI: 10.1145/3567445.3569163. URL: https://doi.org/10.1145/3567445.3569163 (cit. on pp. 18, 30, 40, 50, 51, 56, 59).

[LLG12] H. Lu, J. Li, M. Guizani. "A novel ID-based authentication framework with adaptive privacy preservation for VANETs". In: *2012 Computing, Communications and Applications Conference*. 2012, pp. 345–350. DOI: 10.1109/ComComAp.2012.6154869 (cit. on p. 83).

[Mon23] MongoDB Inc. 2023. URL: https://www.mongodb.com/de-de (cit. on p. 51).

[Pim17] J. R. Pimentel. "Data heterogeneity, characterization, and integration in the context of autonomous vehicles". In: *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*. 2017, pp. 4571–4576. DOI: 10.1109/IECON.2017.8216787 (cit. on pp. 20, 40).

[PKML16] S. Park, J. Kim, R. Mizouni, U. Lee. "Motives and Concerns of Dashcam Video Sharing". In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. CHI '16. San Jose, California, USA: Association for Computing Machinery, 2016, pp. 4758–4769. ISBN: 9781450333627. DOI: 10.1145/2858036.2858581. URL: https://doi.org/10.1145/2858036.2858581 (cit. on p. 40).

[QE16] C. Qin, H. Eichelberger. "Impact-minimizing runtime switching of distributed stream processing algorithms." In: *EDBT/ICDT Workshops*. Vol. 2016. 2016 (cit. on pp. 26, 83, 84).

[Red23] Redis Ltd. 2023. URL: https://redis.io (cit. on p. 51).

[RMVL19] P. H. Rettore, G. Maia, L. A. Villas, A. A. F. Loureiro. "Vehicular Data Space: The Data Point of View". In: *IEEE Communications Surveys & Tutorials* 21.3 (2019), pp. 2392–2418. DOI: 10.1109/COMST.2019.2911906 (cit. on p. 19).

[Sch14] F. M. Schaub. “Dynamic privacy adaptation in ubiquitous computing”. PhD thesis. Universität Ulm, 2014 (cit. on p. 83).

[SÇZ05] M. Stonebraker, U. Çetintemel, S. Zdonik. “The 8 Requirements of Real-Time Stream Processing”. In: *SIGMOD Rec.* 34.4 (Dec. 2005), pp. 42–47. ISSN: 0163-5808. DOI: 10.1145/1107499.1107504. URL: https://doi.org/10.1145/1107499.1107504 (cit. on pp. 21, 59).

[SGB+22] C. Stach, C. Gritti, J. Bräcker, M. Behringer, B. Mitschang. “Protecting Sensitive Data in the Information Age: State of the Art and Future Prospects”. In: *Future Internet* 14.11 (2022). ISSN: 1999-5903. DOI: 10.3390/fi14110302. URL: https://www.mdpi.com/1999-5903/14/11/302 (cit. on pp. 17, 23).

[SS98] P. Samarati, L. Sweeney. “Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression”. In: 1998. URL: https://api.semanticscholar.org/CorpusID:2181340 (cit. on p. 22).

[Ste97] R. Stephens. “A survey of stream processing”. In: *Acta Informatica* 34.7 (July 1997), pp. 491–541. ISSN: 1432-0525. DOI: 10.1007/s002360050095. URL: https://doi.org/10.1007/s002360050095 (cit. on pp. 20, 21).

[The22] The Apache Software Foundation. *Dataflow Programming Model*. 2022. URL: https://nightlies.apache.org/flink/flink-docs-release-1.2/concepts/programming-model.html#dataflow-programming-model (cit. on p. 53).

[The23a] The Apache Software Foundation. 2023. URL: https://flink.apache.org (cit. on p. 50).

[The23b] The Apache Software Foundation. 2023. URL: https://storm.apache.org (cit. on p. 50).

[The23c] The Apache Software Foundation. *Overview*. 2023. URL: https://nightlies.apache.org/flink/flink-docs-master/docs/dev/datastream/operators/overview/#task-chaining-and-resource-groups (cit. on p. 78).

[The23d] The Apache Software Foundation. *Windows*. 2023. URL: https://nightlies.apache.org/flink/flink-docs-release-1.17/docs/dev/datastream/operators/windows/ (cit. on p. 56).

[The23e] The Apache Software Foundation. *Working with State*. 2023. URL: https://nightlies.apache.org/flink/flink-docs-release-1.17/docs/dev/datastream/fault-tolerance/state/#working-with-state (cit. on p. 51).

[Tİ+23] S. Turgay, İ. İlter, et al. “Perturbation Methods for Protecting Data Privacy: A Review of Techniques and Applications”. In: *Automation and Machine Learning* 4.2 (2023), pp. 31–41 (cit. on pp. 22, 23).

[TVC+15] F. Terroso-Sáenz, M. Valdés-Vela, F. Campuzano, J. A. Botia, A. F. Skarmeta-Gómez. “A complex event processing approach to perceive the vehicular context”. In: *Information Fusion* 21 (2015), pp. 187–209. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2012.08.008 (cit. on p. 20).

[Udo20] C. R. Udo Schifferdecker. *High-Performance Computing Platforms in the Automobile Mastering OTA: Automotive and IT domains are converging*. 2020. URL: https://cdn.vector.com/cms/content/products/vconnect/docs/2020-02_Automobil-Elektronik_Mastering-Automotive-OTA.pdf (cit. on p. 27).

[Veca] Vector Informatik GmbH. *Measuring Everything*. Accessed: October 26, 2023. URL: https://cdn.vector.com/cms/content/products/canape/Docs/MeasuringEverything_WhitePaper_EN.pdf (cit. on p. 20).

[Vecb] Vector Informatik GmbH. *MICROSAR Connect | Data Collector*. URL: https://www.vector.com/de/de/products/products-a-z/embedded-components/microsar-connect/vehicle-data-collector/# (cit. on p. 27).

[Ver15] Ververica. *How Apache Flink handles backpressure*. Ed. by U. Celebi. 2015. URL: https://www.ververica.com/blog/how-flink-handles-backpressure (cit. on p. 78).

[WK15] M. A. Will, R. K. Ko. "Chapter 5 - A guide to homomorphic encryption". In: *The Cloud Security Ecosystem*. Ed. by R. Ko, K.-K. R. Choo. Boston: Syngress, 2015, pp. 101–127. ISBN: 978-0-12-801595-7. DOI: https://doi.org/10.1016/B978-0-12-801595-7.00005-7. URL: https://www.sciencedirect.com/science/article/pii/B9780128015957000057 (cit. on p. 22).

[WRW+22] D. Wang, J. Ren, Z. Wang, Y. Zhang, X. (Shen. "PrivStream: A privacy-preserving inference framework on IoT streaming data at the edge". In: *Information Fusion* 80 (2022), pp. 282–294. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2021.11.013. URL: https://www.sciencedirect.com/science/article/pii/S1566253521002384 (cit. on p. 83).

All links were last followed on October 26, 2023.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature