Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master's Thesis Nr. 73

Protecting Private Information in Event Processing Systems

Yves Grau

Course of Study:	Informatik
Examiner:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Supervisor:	Dr. rer. nat. Muhammad Adnan Tariq
Commenced:	July 15, 2015
Completed:	January 14, 2016
CR-Classification:	C.2.1, C.2.4, D.4.6, K.6.5

Abstract

With the increasing number of sensors and smart objects in our daily use, the Internet of Things (IoT) becomes realistic. Thereby, modern applications like “e-health applications” or “smart homes” join our everyday life. These applications have the capability to detect situations of the real world and react to them. *Complex Event Processing* (CEP) systems can detect such occurring situations, which are in the form of event patterns, efficiently.

Besides many benefits which such applications entail, it should not be forgotten that they have a huge impact on privacy. Therefore, it is important that a user has the possibility to decide on his own which complex information he wants to share and which not. This thesis presents a pattern-based access control algorithm which tries to conceal all privacy information in an event stream without destroying the public information. The idea is to reorder a specific set of events of the event stream in such a way that patterns which would result in privacy violations do not longer occur. The evaluation shows that a reorganization of events is possible in many cases without loss of public information.

Kurzfassung

Das Internet der Dinge (InD) wird mit zunehmender Anzahl von Sensoren und „Smart Objekten“ im täglichen Gebrauch immer realistischer. Dadurch erhalten neuartige Anwendungen wie „E-Health Applikationen“ oder „Smart Homes“ Einzug in unseren Alltag. Diese Anwendungen besitzen die Fähigkeit, Situationen aus der realen Welt zu erkennen und entsprechend darauf zu reagieren. *Complex Event Processing* (CEP) Systeme können solche auftretenden Situationen effizient in Form von Ereignismustern erkennen.

Neben den vielen Vorteilen, die solche Anwendungen mit sich bringen, sollte jedoch nicht vergessen werden, dass sie einen immensen Eingriff in die Privatsphäre vornehmen. Daher ist es wichtig, Nutzern die Möglichkeit zu bieten selbst zu entscheiden, welche ihrer komplexen Informationen geteilt werden sollen und welche nicht. Diese Masterarbeit stellt einen musterbasierten Algorithmus vor, welcher versucht alle privaten Informationen in einem Ereignisstrom zu verschleiern ohne dabei die öffentlichen Informationen zu zerstören. Die Idee ist, ausgewählte Ereignisse des Ereignisstroms so umzustellen, dass bestimmte Muster, welche eine Verletzung der Privatsphäre zur Folge hätten, nicht mehr auftreten. Die Evaluierung zeigt, dass in vielen Fällen eine Umstellung von Ereignissen ohne Verlust von öffentlichen Informationen möglich ist.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

The Internet of Things (IoT) becomes more and more important. The number of items and devices which are connected to the internet increases very strongly and this growth is expected to hold on [Int14]. Therefore, security, privacy and scalability is a huge issue in the field of IoT applications. Very powerful systems in the area of IoT are stream processing systems [CM12a], especially complex event processing (CEP) systems.

CEP systems process primitive sensor data event streams to monitor and react to occurring situations in the real world. An abstract architecture of a CEP system is shown in Figure 1.1. A CEP system basically consists of a set of producers, a set of consumers, a set of event definition rules and the CEP middleware. The producers generate primitive events of a certain event type and send them to the CEP middleware. According to the event definition rules, the CEP middleware processes the incoming event stream, generates high-level events and routes these to interested consumers. The consumers react according to the reported information [CM13]. In other words, physical sensors (producers) observe the real world, the CEP middleware detects specific predefined situations and the consumers react to them. For instance, consider a car tracking application as a CEP system. Producers are the sensors in the car which capture information like GPS-position, velocity, oil temperature, tire pressure and other sensors. The CEP middleware can monitor the status of the car. This status will be routed to the consumers, for example a car repair shop or the car insurance. According to the reported status, the car repair shop can order spare parts in advance and make an appointment if a part of the car seems to break down in near future. The car insurance can, for example, adjust

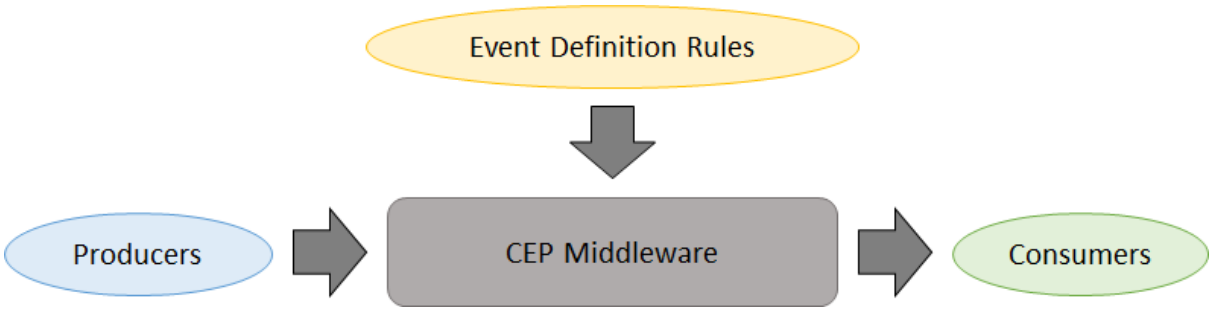


Figure 1.1: Architecture of a CEP system [CM13]

its fees according to the driving behavior of the owner. Consequently, an owner can influence the height of the fee he has to pay, for instance, by the treatment of his car, the frequency and duration of driving and by following the traffic rules.

Along with the increasing number of IoT applications goes the rising amount of revealed critical private data. The example above shows clearly that privacy and security are huge issues in the field of IoT. It is quite easy to see that the fee of the car insurance company will rise if a critical driving behavior is recognized. For that reason, the owner needs a mechanism to restrict the stream of data to the IoT application or at least to the different consumers. Current access control mechanisms mainly focus on restricting the access on specific attributes or flows [CCFT09], [ARX11], [XRAG13], [MPE+10], [SKRR13]. Often, those approaches are not sufficient. Just consider a case where a pattern of ordered event types, and not an attribute value of appearing events, leads to a privacy violation. Because of these privacy violations, there is a need for a *pattern-based access control* mechanism to suppress unintended patterns in the revealed data of the IoT application. All patterns which lead to a privacy violation are called *private patterns* or *concealed patterns*. The patterns which should be revealed to the system, because they capture no critical private information, are called *public patterns*. We talk about a pattern match if the event stream contains this specific public or private pattern.

To the best of our knowledge, there is currently only one solution [WHRN13] challenging the problem of *pattern-based access control*. Their basic idea is to modify the event stream to the CEP middleware in such a way that it contains no private patterns. If a private pattern occurs in the original event stream, their strategy is to suppress one event instance which participates in that specific private pattern. Consequence is that no private pattern remains in the modified event stream and therefore, the CEP middleware cannot reveal the critical private information.

One problem with this approach is that in some cases also public patterns are suppressed. Assume an original event stream which contains a match for a private and a match for a public pattern. Due to the private pattern match, the algorithm suppresses one event instance which participates in that. Therefore, the CEP middleware does not recognize the private pattern and the critical private data stays hidden from the application. But consider the case that the suppressed event instance also participates in the public pattern. This means that the modified event stream does not only lose the information of the private pattern but also the information of the public pattern. The suppression introduces a not matching public pattern. Hence, the IoT application does not get some uncritical privacy information which usually should be revealed. This has undoubtedly a negative effect on the systems Quality of Service (QoS). Another aspect is the very high impact of the suppression approach on the inter arrival rates of events. The inter arrival rate defines the time span from the arrival of an event until the next arrival of such an event of the same type at the system [Raq12]. Consider static inter arrival rates

of events. For instance, an event type has an inter arrival rate of $5ms$. Suppressing one event of this type leads to a doubled inter arrival rate of $10ms$ in the modified event stream. For an attacker it might be possible to detect these anomalies and reproduce the original event stream without suppressed events, by simply observing the event stream.

This thesis proposes a new approach of *pattern-based access control*. The basic idea is to reorder the events of the event stream in such a way that possibly all public patterns are preserved and at the same time all private patterns are concealed. This means that the IoT application still reveals the whole uncritical public information while the critical privacy information is hidden. With the idea of event stream reordering, we see a lot of benefits according to the event stream suppression approach [WHRN13]. One big advantage is that the number of events in the original and in the modified stream are the same. Consider the case from above where the same event type participates in the public and private pattern. If the suppression approach chooses this certain event type for suppression, there is no possibility left to reveal the information of the public pattern to the IoT application. In contrast, the reordering approach has the opportunity to conceal the private pattern while preserving the public pattern in the event stream as long as the patterns are not exactly the same. Therefore, the reordering approach has the potential to reduce the number of public patterns which exists in the original stream but get lost because of the modification. This results in a better QoS of the system.

1.1 Contribution

We developed two reordering algorithms to face the *pattern-based access control* problem. One algorithm is based on a graph. It constructs a directed acyclic graph (DAG) out of the matching public and private patterns. According to the transformation of this graph, the algorithm tries to find a possible reordering of the event stream. The other algorithm is based on an Integer Linear Programming (ILP) formulation. The matching public and private patterns build the constraints of the ILP formulation. As a result, the ILP returns the events which have to be reordered and the corresponding number of time units to move this event on the stream.

The event stream reordering approach has also a high impact on the inter arrival rates of events. Therefore, we introduced an ε-Range for each event type. The reordering algorithm is restricted to reorder an event only in its corresponding ε-Range. The size of this ε-Range mainly depends on the probability distribution according to which this event type is generated. Due to the restriction of this ε-Range, it is not always possible to find a solution in which the modified stream to the IoT application contains no private

pattern. For that reason, we developed a reordering condition to check beforehand if reordering of the event stream is possible or not.

We evaluated both reordering algorithms and compared the results with the event stream suppression approach [WHRN13]. Our focus lay on the number of introduced false-positives and false-negatives. We talk about a false-positive if the modification of the event stream introduces an additional match of a query in the resulting stream. A false-negative describes the other way around. The modification conceals in addition to the private pattern also a public pattern.

1.2 Structure of the Thesis

This thesis is structured as follows:

Chapter 2 – Related Work This chapter introduces fundamental background and presents related work in the area of privacy in event processing systems.

Chapter 3 – System Model and Problem Description This part of the thesis defines the considered underlying system, the event data model, the query model and the selection as well as the consumption policy. In addition to that, it gives a detailed problem description and reveals the problematic properties of the event stream suppression approach [WHRN13].

Chapter 4 – Pattern-based Access Control: Event Stream Reordering Both reordering algorithms, the graph-based algorithm as well as the ILP-based algorithm, are discussed in detail in this chapter. Furthermore, this chapter shows different types of impact which have to be considered of a *pattern-based access control* approach.

Chapter 5 – Evaluation This chapter describes the setup of the evaluation and presents the evaluated results of both developed reordering algorithms as well as the event stream suppression algorithm.

Chapter 6 – Conclusion and Future Work The last chapter gives a summery of this thesis and an outlook on interesting areas for future work.

2 Related Work

This chapter presents the fundamental background of event processing systems. Furthermore, it shows related work in the area of access control and describes the event stream suppression approach [WHRN13] in detail.

2.1 Background of Event Processing Systems

The concept of event processing systems is based on a large number of applications which together need to process different data flows of geographically distributed sources to timely answer complex queries. The reason for the need of event processing systems arises out of the fact that traditional *database management systems* (DBMS) do not fit in the requirements of *timeliness* and *flow processing*. DBMS only process data after it is stored and indexed. But in the concept of event processing is often no need for storing the primitive sensor data because a reported complex event most often contains all the relevant information. For instance, consider a smoke and a temperature sensor for the detection of fire and the notification of a fire alert. If the sensors recognize the complex event *fire* and notify a fire alert, there is no need to store the primitive sensor data which led to the complex event [CM12b].

Many researchers with different background addressed the above problem of DBMS. Therefore, a lot of solutions and models of different types exist. The most popular models are the *data stream processing model* [BBD+02] and the *complex event processing model* [Luc01]. The model of data stream processing evolves from the area of DBMS and presents the type of *Data Stream Management Systems* (DSMS). DSMS differ a lot of traditional DBMS. DBMSs work on persistent data with infrequently updates. A query is executed only once and the corresponding answer is returned. Figure 2.1 shows a model of a DSMS. DSMSs assume different streams of input data. In contrast to DBMS, the DSMS works with many persistent queries which get answered as soon as data arrives. The answers build the output stream. The *store* holds data which might change and is needed to answer queries in future. Data which is useful to answer a query but is not needed necessarily is stored by the *scratch*. The *throw* only indicates that unnecessary data gets removed. Intention of DSMS is to update query answers as soon as they

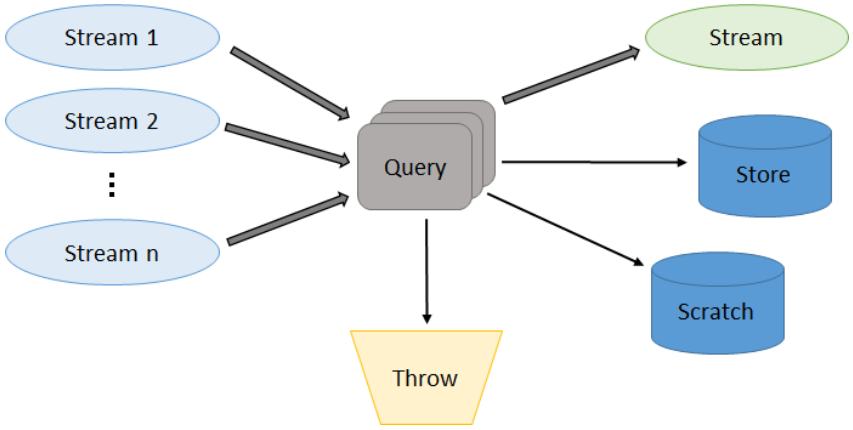


Figure 2.1: Model of a Data Stream Management System[BW01]

change because of new arrived data. Most often, the detection of sequences, ordering relations or patterns [CM12b] is not considered of DSMS. This is the reason why this thesis focuses on the second model, the complex event processing model. Chapter 1 already introduced the model in Figure 1.1 and explained the concept of CEP. In contrast to the data stream processing model, this model assumes streams of primitive events, observed by different real world sensors. Operators process the primitive event streams to generate high-level events which represent complex situations of the real world. Furthermore, CEP systems can detect sequences, ordering relations or patterns of events in the event streams [CM12b]. The next section shows how such high-level events, also called complex events, can be specified.

2.2 Event Specification Languages

The need for event specification languages is easy to see. Just consider the already mentioned example of the detection of the complex event *fire*. The combination of events, generated by a temperature and a smoke sensor, can lead to the detection of fire. Furthermore, it is necessary to specify the complex event *fire* according to the raw events of the sensors. Besides, it is possible to describe the high-level event *fire* in many different ways. Below three descriptions are shown [CM10].

- If a smoke event and a temperature event with a value higher than 45 degrees occur within 3 minutes, there is fire [CM10].
- If the average temperature is higher than 45 degrees within the last 3 minutes and a smoke event occurs, there is fire [CM10].
- If a smoke event and at least 10 temperature events with increasing values occur within 3 minutes, there is fire [CM10].

These descriptions already show that an event specification language must have the capability to express different constructs like selections or aggregations. Many different event specification languages are developed [CM10], [ABW06], [BTW+06]. But the *Trio-based Event Specification LAnguage* (TESLA) [CM10] seems to be the most expressive one. Therefore, this section looks into detail of TESLA.

In TESLA an event consists of a certain type which defines the set of attributes of this event. For instance, an event of type *Temp* has an attribute *Room* which defines the location of the measurement and an attribute *Value* which contains the measured temperature value. An example of this event type can have the following form [CM10]:

$$Temp@5(Room = \text{ı}Room1\text{ȷ},\ Value = 20)$$

This temperature event is generated at time 5, in *Room1* with a measured temperature value of 20. TESLA uses so called *TESLA rules* to define the high-level events out of the basic events. A TESLA rule has the following structure [CM10]:

$$\begin{array}{ll} define & CE(Att_1 : Type_1,\ \ldots,\ Att_n : Type_n) \\ from & Pattern \\ where & Att_1 = f_1,\ \ldots,\ Att_n = f_n \\ consuming & e_1,\ \ldots,\ e_n \end{array}$$

The first line defines the resulting complex event with a set of attributes. The *Pattern* of the second line defines the basic events which have to occur for the generation of the complex event. The third line specifies, according to some predefined functions, the values of the attributes in the complex event. The last line determines whether the generation consumes basic events or not. If a basic event is consumed, it cannot participate in the generation of further complex events [CM10]. With this structure it is possible to formulate the example above of the complex event *fire*. As already mentioned, there are many possibilities for a definition. Thus, also this TESLA rule is only one possibility [CM10]:

$$\begin{array}{ll} define & Fire(Value) \\ from & Smoke()\ and \\ & each\ Temp(Value > 45)\ within\ 3min\ from\ Smoke \\ where & Value = Temp.Value \end{array}$$

This TESLA rule creates an complex event *Fire* with one attribute *Value*. The complex event is generated for every temperature event with a value higher than 45 which occurs within 3 minutes after a *Smoke()* event. The attribute value of the complex event is equal to the value of the temperature event. The generation does not consume any primitive events [CM10]. Because of the huge number of sensors it is important to have an efficient detection of such complex events like the *Fire* event. Therefore, the next section shows a strategy for *parallel complex event processing*.

2.3 Parallel Complex Event Processing

The timely detection of complex events is crucial for CEP systems. Otherwise, the applications cannot react in time on occurring situations. For instance, to make an early emergency call it is important to detect the fire as soon as possible. Because of the increasing number of sensors, timeliness becomes very hard to achieve. Therefore, Mayer et al. [MKR15] presents a strategy for parallel complex event processing. One issue in terms of parallelization is to split the stream of events in such a way that no patterns which lead to a complex event get lost. According to this problem, they propose a model for pattern-sensitive stream partitioning. Idea is to split the stream by selections. A selection contains all events which could participate in a pattern of a complex event. After that it is possible that different operators process the selections parallel without loosing any occurring complex events [MKR15].

2.4 Privacy in Event Processing Systems

This section presents approaches in the area of privacy and security in event processing systems. Most of the existing approaches focus on access control and information flow control [CCFT09], [ARX11], [XRAG13], [MPE+10], [SKRR13]. However, the event stream suppression approach [WHRN13] considers the issue of *pattern-based access control*.

2.4.1 Access and Information Flow Control

Schilling et al. [SKRR13] show an access control mechanism which holds over multiple processing steps in the CEP system. Operators are only getting access to event attributes if they fulfill the corresponding access requirements. Access policies for the event attributes assure the inheritance of the access requirements for further operators in the

processing chain [SKRR13]. The access control mechanism of Cao et al. [CCFT09] is role-based and works at query definition time. As soon as a query is submitted, their algorithm checks if the user has the required access privileges. According to the access privileges of the user, the system denies, partially grants or totally grants the submitted query. If the query is only partially granted, the system rewrites the query in such a way that the response contains only authorized data [CCFT09].

Adaikkalavan et al. [ARX11] and Xie et al. [XRAG13] present a flow control mechanism which is based on a set of security levels. They classify the users and the information in different security levels. In [ARX11], these security levels participate in a security structure which defines a partial order between the different levels. The idea is to allow the information flow only from dominated to dominating security levels. Therefore, information of a certain security level can never reach a security level which has lower security privileges [ARX11]. The approach of Migliavacca et al. [MPE+10] is also an event flow control mechanism. But instead of security levels they use security tags. Every event flow is forced to have a security tag. Events can just flow to processing units with higher security privileges than the associated tag [MPE+10].

2.4.2 Pattern-based Access Control

Wang et al. [WHRN13] focus on *pattern-based access control*. In many situations pure access control is not enough to hide all the critical private data. Just consider the case where a pattern of occurring event types leads to a privacy violation. These patterns are called *private patterns*. For instance, such a private pattern can be the following [WHRN13]:

$$P_1 : \ SEQ(Exit - patient - room, \ Enter - psychiatrist - office) \\ within \ 5 \ min$$

With such a pattern in the event stream it is possible to reveal the information that this specific patient has psychic problems. The first intention of Wang et al. is to only report the public patterns. Figure 2.2 shows that this idea cannot solve the problem. The system reports a match of query $Q_1 : \ SEQ(Exit - patient - room, \ Sanitize)$ and query $Q_2 : \ SEQ(Wash, \ Enter - psychiatrist - office)$. With this reported information an attacker can infer that the event stream has also a match of P_1 [WHRN13]. Therefore, they present the event stream suppression approach. The idea is to suppress events of the event stream in such a way that possible private patterns cannot have a match. Suppression of one event of the private pattern in Figure 2.2 will ensure that either Q_1 or Q_2 loses the match. It is always a trade-off between revealing all public patterns and concealing all private patterns [WHRN13].

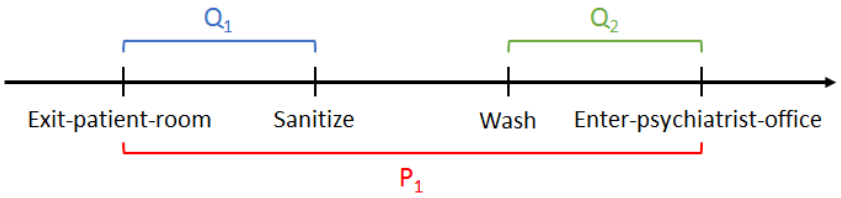


Figure 2.2: An attacker can infer that P_1 has a match because of the matches of Q_1 and Q_2 [WHRN13].

Wang et al. [WHRN13] present two algorithms to address the issue of *pattern-based access control*. The first one is the *optimal type-level algorithm*. This algorithm is based on an ILP formulation which returns the corresponding event type for suppression. $\Sigma = \{E_i\}$ is the set of event types. $x_i \in \{0, 1\}$ are the decision variables whether to suppress $(x_i = 0)$ the event type E_i or not $(x_i = 1)$. The variables $y_j \in \{0, 1\}$ show whether the corresponding query $Q_j \in Q$ is preserved $(y_j = 1)$ after suppression or not $(y_j = 0)$. Same holds for the variables $z_k \in \{0, 1\}$ and the corresponding private pattern $P_k \in P$. The objective function looks as follows [WHRN13]:

$$\text{maximize} \quad \sum_{Q_j \in Q} w(Q_j) N_T(Q_j) y_j + \sum_{P_k \in P} w(P_k) N_T(P_k) z_k$$

The function $w(Q_j)$ represents the utility weight and the importance of a query Q_j. The weight of a private pattern is always negative for indicating the utility loss. $N_T(Q_j)$ gives the expected number of matches of a query Q_j over a period of time T. Goal of the objective function is to maximize the overall utility. $\sigma(Q_j)$ defines the multi-set of event types of a query Q_j. Then the constraints which consider the queries have this form [WHRN13]:

$$0 \leq y_j \leq \frac{1}{|Q_j|} \sum_{E_i \in \sigma(Q_j)} x_i$$

A query is preserved $(y_j = 1)$ if none of the participating event types is suppressed. And a private pattern is concealed $(z_k = 0)$ if one of the participating event types is suppressed. Thus, these constraints look like this [WHRN13]:

$$1 \geq z_k \geq \frac{1}{|P_k|} - 1 + \frac{1}{|P_k|} \sum_{E_i \in \sigma(P_k)} z_i$$

The ILP returns the event types which should be suppressed to reach the highest utilization.
The second algorithm of Wang et al. [WHRN13] is the *hybrid instance-level algorithm*. Problem of the algorithm above is that it uses the expected number of pattern matches

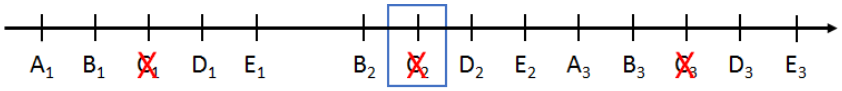


Figure 2.3: Sub-optimal type-level solution [WHRN13].

$N_T(Q_j)$. This is only an average value and because of fluctuations it is possible that this value changes and the algorithm gets sub-optimal. Figure 2.3 shows such a case. Assume this private pattern $P_1 : SEQ(A, C, E)$. According to the solution of the ILP, the algorithm suppresses all events of type C. Due to the missing of event A_2 there is no need to suppress C_2. If a query exists which would have a match if the suppression of C_2 is prevented, then the type-level solution is sub-optimal. Therefore, the hybrid-instance-level algorithm decides the suppression for every occurring event instance. For an arrived event, the algorithm takes the partial match of a query into account and estimates on the basis of the arrival times or some periodicity whether the query will get a match or not. The algorithm does the same for the private pattern by comparing the expected utility gain of the queries and the expected utility penalty of the private patterns. If the expected utility penalty is higher than the utility gain, the algorithm suppresses the considered event instance. This algorithm will not suppress event C_2 of Figure 2.3 because, by checking the partial match of the private pattern, the algorithm recognizes that the private pattern cannot occur because of the absence of event A_2 [WHRN13].

3 System Model and Problem Description

This chapter introduces the basic system model with the used event data and query model. Moreover, it gives a detailed description of the problem and shows the main drawbacks of the event stream suppression approach [WHRN13].

3.1 System Model

The underlying CEP system consists of a set of connected distributed nodes. Producers, consumers and operators are deployed on these nodes. The CEP system and its behavior can be modeled by an operator graph. Figure 3.1 shows such an operator graph. A set of producers generate different types of primitive events and send them to the CEP middleware. Usually a producer is any type of sensor, for instance a temperature sensor. These sensors observe the real world and generate corresponding events. A generated

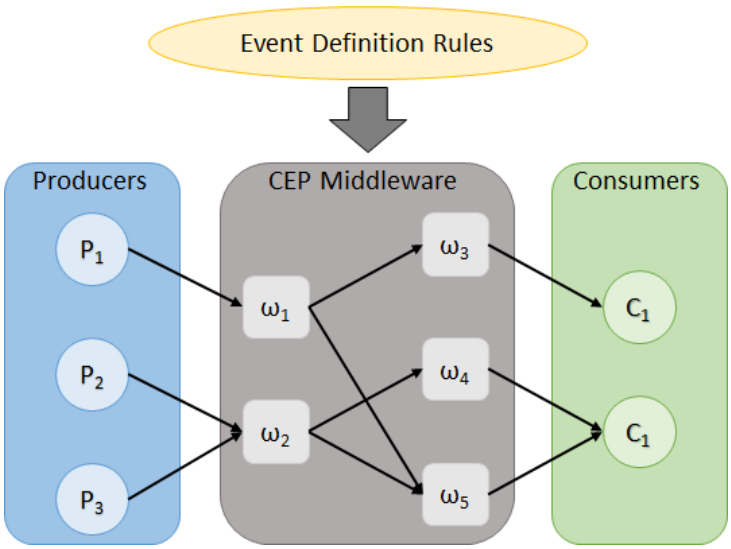


Figure 3.1: Model of a CEP system with an operator graph consisting of 3 producers, 5 operators and 2 consumers. The event definition rules specify the resulting high-level events out of the incoming event streams.

primitive event is an instance of one specific type. It consists of a sequence number and a set of attributes which are defined by the corresponding event type. In terms of such a temperature sensor, the event is, for example, of type *temperature* and consists of a *temperature value* and a *timestamp*.

The CEP middleware is built up of a set of interconnected operators which processes the incoming event streams and generates high-level events according to some predefined rules. An operator is one specific processing node in the CEP middleware. Its job is to process and transform the incoming event stream according to predefined rules to one or many outgoing event streams. These outgoing event streams are connected either to another processing operator or to a consumer. In the area of CEP a number of different operator types exists, for instance [CM12b]:

- Sequence: The *sequence operator* considers the ordering relations of incoming events. If the predefined order of events occurs in the incoming event stream, then we call that a match of the sequence operator.
- Selection: The *selection operator* filters the event stream according to a defined event type or an attribute value.
- Aggregation: The *aggregation operator* combines multiple events. For example, by calculating an average value of one attribute.
- Negation: The *negation operator* considers the non-appearance of events in the event stream.

The cooperative processing of the different operators results in a high-level event. For instance, a high value of a temperature sensor and the alarm of a smoke sensor can result in a high-level event like the detection of fire [CM13]. In this thesis we only focus on the sequence operator.

An operator does not check the whole event stream for a match at once. Usually only small parts are considered by an operator. The size and location of this parts in the event stream are defined by the *window*. The size of a window can either be *time-based* or *count-based*. A time-based window considers all the arrived events in a specified period of time, for instance all arrived events in the last 2 minutes. In contrast, a count-based window contains only a defined number of events, for example the last 20 arrived elements of the event stream [CM12b]. The characteristics of a *sliding-window* define the location and movement on the event stream. As soon as a new event arrives or some time threshold exceeds, the sliding-window increases its lower and upper bound and moves one step further on the event stream [GÖ03]. This thesis uses a *time-based sliding-window*. The window contains all events which arrive in a specified period of time. The next window starts at the end point of the previous window.

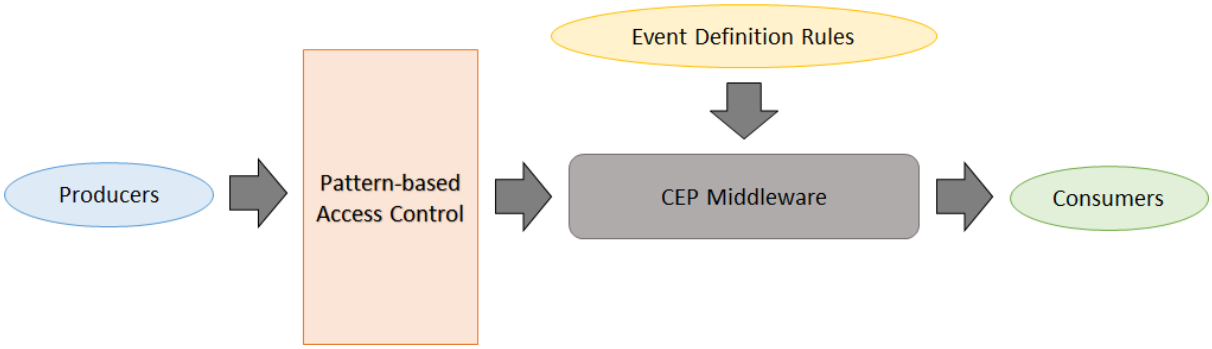


Figure 3.2: Simplified model of a CEP system with the additional *Pattern-based Access Control* component.

A *pattern* defines the exact ordering relations for a set of events. A sequence operator checks the match of such patterns in a window. For instance, a pattern can define the ordering relation that an event of a temperature sensor has to occur before an alarm event of a smoke sensor in the event stream. As already mentioned in chapter 1, security and privacy play a huge role in event processing systems. For this reason, we introduce two types of patterns, the *public patterns* and the *private patterns*. A public pattern is a pattern which should be revealed to the IoT application because it only contains uncritical privacy information. In contrast, the private pattern or concealed pattern is a pattern which deals with critical privacy information and should be hidden from the application. We call a private pattern *hard-constraint* if absolutely no match is allowed to be revealed [WHRN13]. *Soft-constraint* private patterns give the flexibility that private pattern matches are not strictly forbidden, for example, if the suppression leads to the not matching of very important public patterns. This property enables the algorithm the opportunity to have a tradeoff between revealing important public patterns and suppressing private patterns [WHRN13].

Finally, the CEP middleware routes these high-level events to interested consumers which can react to the reported situations. According to a detected fire, the consumer can make, for example, an emergency call.

The approach of this thesis adds a new component between the producers and the CEP middleware into the already presented model. Figure 3.2 shows the already presented system model with this additional *Pattern-based Access Control* component. Main difference of the model is that the produced event stream first goes through the *Pattern-based Access Control* component before it reaches the CEP middleware. Therefore, the original event stream is only visible at the *Pattern-based Access Control* component. The modified event stream is routed directly to the CEP middleware. We assume that the event streams of the different producers are merged together into one input event stream.

3.1.1 Event Data Model

An event represents an observation of the real world. Different producers generate different types of events. The type defines the set of attributes in an event instance. Additionally, all events contain a sequence number. The time of arrival at the first component of the system defines a total order between all event streams of the different producers. For instance, an event can have the following structure:

$$Distance(seq = 1, value = 5)$$

This is an event produced by a distance sensor with a sequence number of 1 and a measured distance of 5 meters.

For the following examples this thesis uses a more abstract notation. We consider a set of available event types $\Sigma = \{A, B, C, D, ...\}$. An event of type A with a sequence number of 1 is written by:

$$A_1$$

With this abstract notation an event stream can look like the one in Figure 3.3. The order according to the arrival time is given from left (early) to right (late).

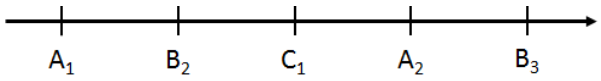


Figure 3.3: Example event stream.

3.1.2 Query Model

A *query* represents a request to the event processing system. It precisely defines the situation or rather the information, the IoT application is interested in. As already mentioned, in this thesis we only focus on the sequence operator [CM12b]. Therefore, we also consider only sequence queries. A sequence query consists of exact one pattern which defines the high-level event and has the following form [WHRN13]:

$$Q = SEQ(Smoke\_Alarm,\ High\_Temperature\_Sensor\_Value)$$

This sequence query is interested in the *Smoke_Alarm* event and the *High_Temperature_Sensor_Value* event. For simplicity, we use the *High_Temperature_Sensor_Value* event which represents a temperature sensor event with an high attribute value of the measured temperature. Moreover, the query defines the ordering relation that the *Smoke_Alarm* event has to occur before the *High_Temperature_Sensor_Value* event. If this sequence query finds a match on an event stream, the resulting high-level event might be the

detection of fire [CM13]. Figure 3.4 shows such an event stream with a match of the sequence query. Each of the shortcuts represents an event and has the following meaning: LTSV = *Low_Temperature_Sensor_Value*, HTSV = *High_Temperature_Sensor_Value*, SA = *Smoke_Alarm*. We assume a window which contains all four events. It is easy to see that the sequence query has a match because both necessary event types participate in the event stream and the ordering relation is also fulfilled the *Smoke_Alarm* event (second position) occurs before the *High_Temperature_Sensor_Value* event (forth position).

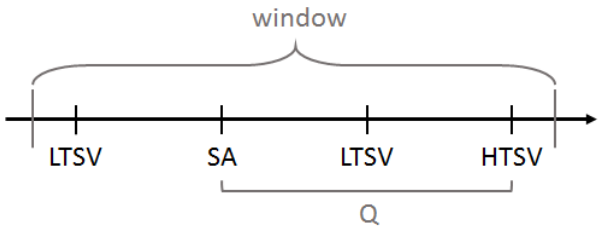


Figure 3.4: An event stream with a match of query Q.

An example for an event stream with no match of the sequence query is shown in Figure 3.5. Again, the window contains all four events. Both necessary event types participate in the event stream, but the defined ordering relation of the sequence query is not fulfilled. Therefore, the sequence query does not match.

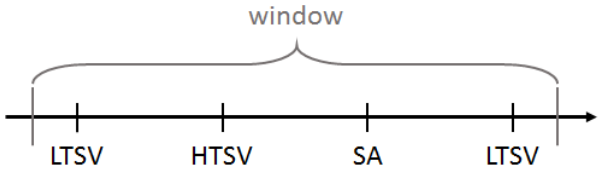


Figure 3.5: An event stream without a match of query Q.

For the query model we also use a more abstract notation. We consider the same set of event types $\Sigma = \{A, B, C, D, ...\}$ like in section 3.1.1. An ordering relation is written as $or(A, B)$ which defines that event A has to occur before event B. A sequence query which considers the event types $\{B, D, E\}$ and the ordering relations $or(D, B)$, $or(B, E)$ and due to transitivity also $or(D, E)$, has this form:

$$Q = (D, B, E)$$

In terms of privacy we differentiate between queries and private patterns. Queries consist of public patterns and should be revealed to the IoT application. In contrast, Private patterns are defined exact like queries but handle critical privacy information and should be suppressed. In notation we distinguish them by denoting all queries by $Q_i = (...)$ and all private patterns by $P_i = (...)$.

3.1.3 Selection Policy

In a window with multiple instances of the same event type, the *selection policy* defines which event type to select for the query match [CM12b]. For instance, consider the following query:

$Q = (A, B)$

Figure 3.6 shows the given input event stream. Assume that the window contains all six events of the input stream. It is easy to see that several combinations exist to achieve a match of the query. Overall, three combinations are possible:

(A_1, B_1)

(A_2, B_1)

(A_3, B_1)

The selection policy used in this thesis takes always the first occurrence of the specific event type needed by the query. Therefore, the query in the given example selects the first occurrence of event type A and out of that position the first event of type B. This results in one match of the query with the combination (A_1, B_1).

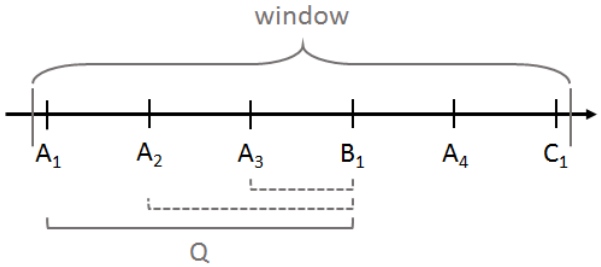


Figure 3.6: An event stream with several possible combinations for a match of query Q.

3.1.4 Consumption Policy

The *consumption policy* defines if an event instance of the event stream can be used multiple times in different query matches [CM12b]. In this thesis the used consumption policy specifies that an event instance can only be used once by a particular query. That means, different queries can use the same event instance in one window. Consider the following example with the two queries:

$Q_1 = (A, B)$

$Q_2 = (A, C)$

The window with the event stream is shown in Figure 3.7. Due to the presented selection and consumption policy, the sequence operator reports the following two matches of query Q_1:

(A_1, B_1)

(A_2, B_2)

An additional match of Q_1, for example, like the combination (A_3, B_2) is not possible because B_2 is already used for the match (A_2, B_2). The considered window only contains one event instance of type C. That is why query Q_2 only results in one match: (A_1, C_1).

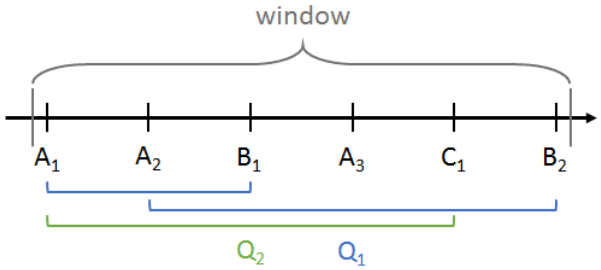


Figure 3.7: An event stream with two matches of Q_1 and one match of Q_2.

3.2 Problem Description

This thesis addresses the problem of preventing the event processing system from revealing critical privacy information in terms of patterns. Given is a set of event types, a set of queries and a set of private patterns. In this thesis we only consider cases with one private pattern. Goal is to modify the original event stream in such a way that all private patterns are concealed (hard-constraint) or at least the number of occurring private patterns is minimized (soft-constraint). Furthermore, the modification should

not have a high impact on the QoS in terms of *false-negatives* and *false-positives*. We talk about a false-negative if a matching query of the original event stream does not have a match anymore after the modification. Thus, uncritical public information is lost because of the modifications. A false-positive defines the other way round, a query has a match in the modified event stream but has no match in the original stream. In this case, the algorithm generates information and situations which did not occur in the real world. Both are very important cases and an algorithm should try to avoid them.

The event stream suppression approach [WHRN13] already addresses this problem. Their idea is to suppress one event instance which participates in the matching private pattern. One drawback of this approach is that it might lead to a high number of false-negatives. As soon as the suppressed event instance also participates in a matching query, this query will not match in the modified event stream and consequently, a false-negative is introduced. The following example shows this case. Consider this available set of different event types $\Sigma = \{A, B, C, D, E\}$. Additionally, we assume the following queries and the private pattern:

$Q_1 = (B, C, D)$ $\quad$ $P_1 = (A, C, E)$

$Q_2 = (B, E)$

$Q_3 = (A, D)$

Figure 3.8 shows the window with the input event stream. In the window, all queries and the private pattern have a match. Therefore, the event stream suppression approach will suppress one event instance which participates in the private pattern.

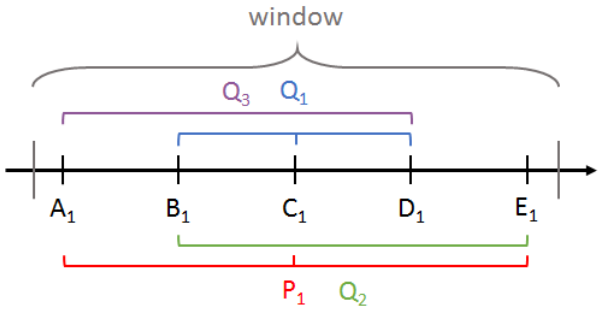
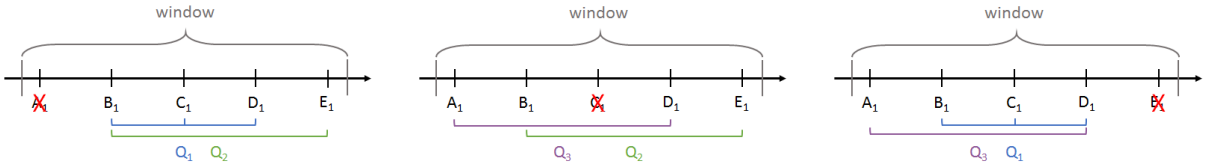


Figure 3.8: An event stream with a match of the queries Q_1, Q_2 and Q_3 as well as a match of the private pattern P_1.

Figure 3.9 shows the three possible modified event streams. In the first modified event stream (Figure 3.9a) event A_1 is suppressed, in the second (Figure 3.9b) C_1 and in the third (Figure 3.9c) E_1. None of the modified event streams contain the private pattern but problem in this example is that the matching queries contain all event types of the



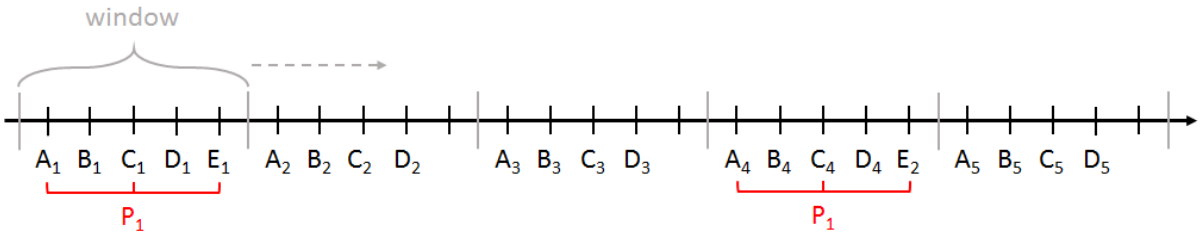
(a) Suppression of event A_1. **(b)** Suppression of event C_1. **(c)** Suppression of event E_1.

Figure 3.9: The three possible modified event streams of the event suppression approach. Each of them introduces a false-negative.

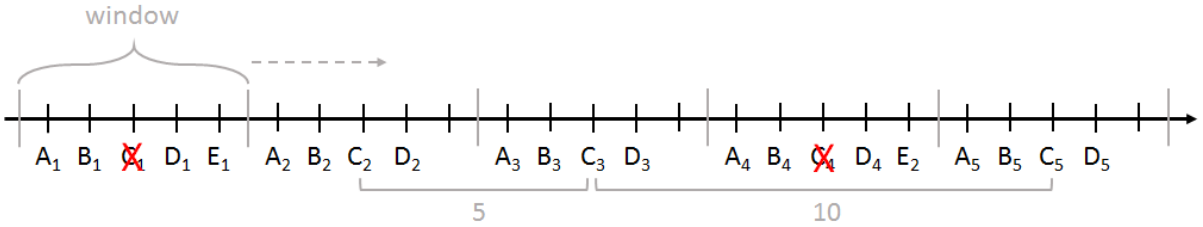
private pattern. Event type A participates in Q_3, C in Q_1 and E in Q_2. Suppressing an event instance of one of these event types leads to a false-negative. The first modified event stream does not have a match of Q_3, the second no match of Q_1 and the third no match of Q_2.

There is a second drawback of the approach because of the high impact on the inter arrival rates. Suppression of one event instance of a certain type leads to a doubled inter arrival rate. By observing the modified event stream, an attacker might detect these anomalies and reproduce the original event stream without suppressed events. The following example shows this problem in detail. Consider the same event types, the same queries and the same private pattern like in the example above. The different event types have these static inter arrival rates: $(A, 5)$; $(B, 5)$; $(C, 5)$; $(D, 5)$; $(E, 15)$. The given event stream is shown in Figure 3.10a. The first and the forth window contain a private pattern match. We assume that the event stream suppression approach [WHRN13] suppresses event instance C_1 in the first window and C_4 in the forth window. In Figure 3.10b is the modified event stream. The modified event stream shows these anomalies where the suppressed event type C has a doubled inter arrival rate. Between the event instances C_3 and C_5 is a doubled inter arrival rate of 10. In contrast, between C_2 and C_3 is the defined inter arrival rate 5.

This thesis proposes a new approach of *pattern-based access control*. The idea is to reorder the original event stream in such a way that all private patterns are concealed. More formally, the reordering approach gets a set of event types with different dynamic inter arrival rates. Additionally, every considered window gives two different sets of ordering relations. The first set of ordering relations is given by the queries which have a match in the considered window and the second set results out of the ordering relations of the matching private patterns. Goal is to conceal the private patterns while the number of introduced false-negatives and false-positives should be minimized. Furthermore, the reordering approach is restricted to reorder an event instance in the corresponding ε-Range. The ε-Range depends on the probability distribution and its associated variance [Weid]. Purpose of the ε-Range is to conceal the impact of the reordering approach on the inter arrival rates.



(a) An event stream with two private pattern P_1 matches in the first and the forth window.



(b) Modified event stream with suppressed event C_1 and C_4. P_1 does not have a match.

Figure 3.10: Input (a) and result stream (b) of the event stream suppression approach.

The following example shows the benefits of the reordering approach towards the event suppression approach. For this example we also consider the set of event types, the queries and the private pattern from the example above. The event stream looks exactly like in the example of the suppression approach in Figure 3.8. As already mentioned, the suppression approach introduces a false-negative. In contrast, the reordering approach can conceal the private pattern while preserving all matching queries. The modified event stream after reordering is shown in Figure 3.11. All queries still match in this modified event stream, but the considered private pattern has no match anymore. Therefore, the reordering approach can achieve a lower number of false-negatives which results in a better Quality of Service.

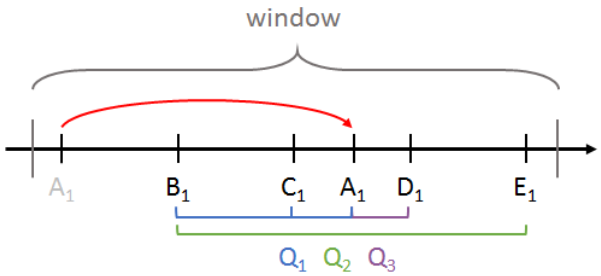


Figure 3.11: Modified event stream of the reordering approach with a match of the queries Q_1, Q_2 and Q_3. No false-negative is introduced.

4 Pattern-based Access Control: Event Stream Reordering

This chapter gives a detailed description of the event stream reordering approach. First of all, it presents fundamental aspects of the *pattern-based access control*. Furthermore, this chapter shows different types of inter arrival rates and their influences on a reordering solution. It describes the ε-Range and its intention in detail. At the end of this chapter, the two event stream reordering solutions are presented.

We developed two different types of reordering solutions. The first algorithm constructs a directed acyclic graph (DAG) out of the matching queries and private patterns. The event instances represent the nodes and the ordering relations build the edges of the graph. The algorithm transforms the generated graph to achieve a reordered event stream with no private pattern matches. The second solution is based on an Integer Linear Programming (ILP) formulation. The constraints consist of the ordering relations of the matching queries and private patterns. The ILP returns the events for reordering and the corresponding time differences to move these events on the event stream.

4.1 Overview

The goal of the event stream reordering is to conceal the critical privacy information while preserving the uncritical public information. In addition, the number of introduced false-negatives and false-positives should be minimal. Another aspect concerns the impact on the inter arrival rates. For an attacker, it should not be possible to discover the events with changed inter arrival rates. Otherwise, it also might be possible to rebuild the original event stream and thus, to reveal the critical privacy information.

With the following example (slightly modified [WHRN13]) we describe the basic principles of the event stream reordering approach. Assume this set of available event types $\Sigma = \{A, B, C, D, E\}$. The following queries and the private pattern are given to the system:

$Q_1 = (B, C, D)$ $\quad$ $P_1 = (A, C, E)$

$Q_2 = (D, E)$

$Q_3 = (E, A)$

Figure 4.1 shows the input event stream. The window contains all five event instances. The queries Q_1 and Q_2 and the private pattern P_1 have a match in the window. Therefore, the event stream reordering approach has to reorder the event stream in such a way that P_1 is hidden while the matching queries Q_1 and Q_2 are preserved. For a solution, the reordering algorithm has to face two problems. The first problem is to find a set of events to reorder. The second problem is to find the positions where the events should be placed on. The following sections look at these two problems in detail.

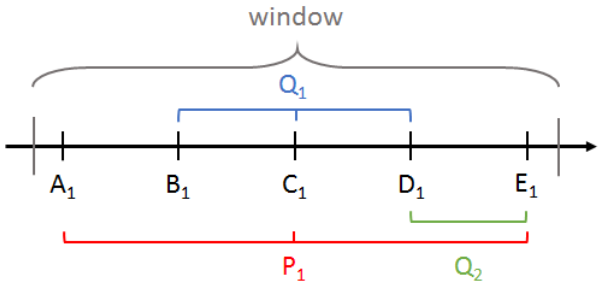


Figure 4.1: Input event stream with matching queries Q_1 and Q_2 as well as a match of the private pattern P_1.

4.1.1 Impact of Potential Events for Reordering

Concerning the first problem, to find a set of events to reorder, take a look at the given ordering relations of the example above in section 4.1. These are the ordering relations of the matching queries and private pattern:

$Q_1 \Rightarrow or(B, C);\ or(C, D)$

$Q_2 \Rightarrow or(D, E)$

$P_1 \Rightarrow or(A, C);\ or(C, E)$

It should be clear that the input event stream fulfills all these ordering relations, otherwise the corresponding query or the private pattern has no match. The ordering relations, defined by the matching queries, should be preserved so that the queries still match in the modified event stream. But if one ordering relation of the private pattern is not fulfilled in the modified event stream, the private pattern is concealed. Thus, the

reordering approach has to negate one of the ordering relations of the private pattern P_1. There are two possible ways to negate an ordering relation. For instance, to negate the ordering relation $or(A, C)$ we can either delay A to a position after C or move C to a position before A. Thus, the events participating in this ordering relation define the events which should be reordered. The decision of choosing one of the ordering relations of the private pattern is not easy and has a huge impact on the result.

Assume the algorithm decides to negate the ordering relation $or(A, C)$. With this decision, the first mentioned problem is solved because A and C are the events to reorder. A possible modified event stream with the negation of ordering relation $or(A, C)$ can have the form shown in Figure 4.2. The event instance A_1 is delayed to the position after event instance C_1. Therefore, the ordering relation $or(A, C)$ is not fulfilled anymore and the private pattern has no match in the modified stream. The queries Q_1 and Q_2 have a match while Q_3 still has no match like in the original event stream. The reordering produced neither a false-negative nor a false-positive. This is the best solution in terms of false-negatives and false-positives a *pattern-based access control algorithm* can achieve.

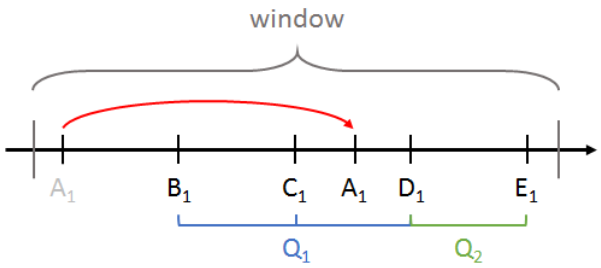


Figure 4.2: Modified event stream with delayed event instance A_1. The private pattern P_1 is concealed.

But now, consider the algorithm decides to negate the ordering relation $or(C, E)$ instead of $or(A, C)$. Then a modified event stream can look like the one in Figure 4.3. To negate the ordering relation $or(C, E)$, event instance C_1 is delayed to the position after event instance E_1. Thus, one of the ordering relations of P_1 is not fulfilled and therefore, P_1 has no match anymore. The query Q_2 has a match and Q_3 still has no match. Up to this point, everything is the same like before, but because of the delayed event instance C_1 also the ordering relation $or(C, D)$ of Q_1 does not apply in the modified event stream. Therefore, the reordering approach introduces a false-negative because query Q_1 has no match anymore.

This small example shows already that the chosen events for reordering have an impact on the number of false-positives and false-negatives. The negation of ordering relation $or(C, E)$ introduces a false-negative. With a deeper look at the matching queries and their ordering relations it becomes clear why this is the case. Due to the transitivity,

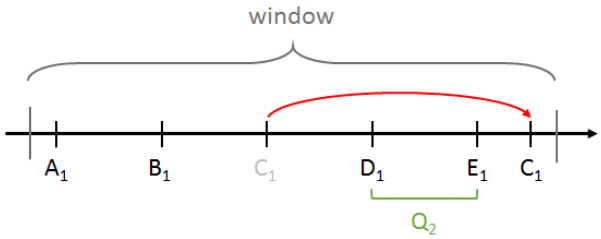


Figure 4.3: Modified event stream with delayed event instance C_1. The private pattern P_1 is concealed. Not matching Q_1 introduces a false-negative.

the ordering relations of the matching queries Q_1 and Q_2 build this additional ordering relation:

(4.1) $or(C, D) \wedge or(D, E) \Rightarrow or(C, E)$

The matching queries Q_1 and Q_2 also define the ordering relation $or(C, E)$. In the example above the algorithm chooses $or(C, E)$ for negation. The expression 4.1 shows that the negation of $or(C, E)$ will result in the negation of either $or(C, D)$ or $or(D, E)$. Which of these ordering relations are affected depends on the positions where the events are placed on. The impact of these positions is shown in the next section 4.1.2. In the example above $or(C, D)$ is affected because event instance C_1 is delayed to the position after event instance E_1. But it is also possible to move E_1 to a position before event C_1. Figure 4.4 shows the resulting modified event stream. In this case, the other ordering relation $or(D, E)$ of expression 4.1 is affected. Difference of this modified event stream and the solution from above is that here query Q_1 has a match and Q_2 has no match. The overall number of false-negatives and false-positives stays the same. The decision to negate the ordering relation $or(C, E)$ was not as good as negating ordering relation $or(A, C)$. Because the negation of $or(A, C)$ introduces neither a false-negative nor a false-positive.

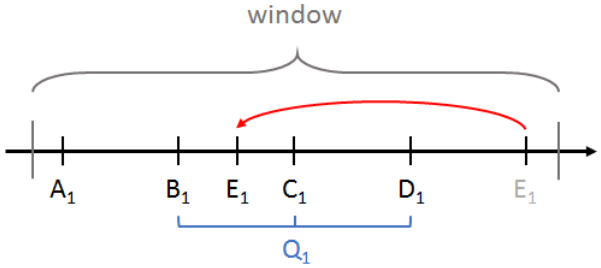


Figure 4.4: The modified event stream with event E_1 moved to the position before C_1. P_1 is concealed. A false-negative is introduced because Q_2 has no match.

In the event stream suppression approach [WHRN13] it is impossible that the algorithm introduces a false-positive as long as a query does not contain a negation, but event negation is not supported. In contrast, the reordering approach can lead to false-positives in the modified event stream. The following example shows this case. We slightly modify the currently used queries and private pattern of section 4.1. Now, they look as follows:

$Q_1 = (B, C, D)$ $\quad$ $P_1 = (A, E, C)$

$Q_2 = (D, E)$

$Q_3 = (E, A)$

The given input event stream is shown in Figure 4.5. The window contains all five event instances. Only query Q_2 and the private pattern P_1 have a match.

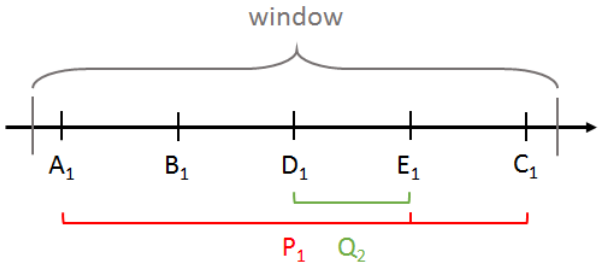


Figure 4.5: Input event stream with a match of Q_2 and P_1.

Assume the reordering approach takes the ordering relation $or(A, E)$ of the private pattern for negation. After reordering the result might be this modified event stream of Figure 4.6. As expected, the generated event stream contains no match for the private pattern P_1. The match of query Q_2 is preserved but additionally, the event stream contains a match for Q_3. The modification introduced a false-positive.

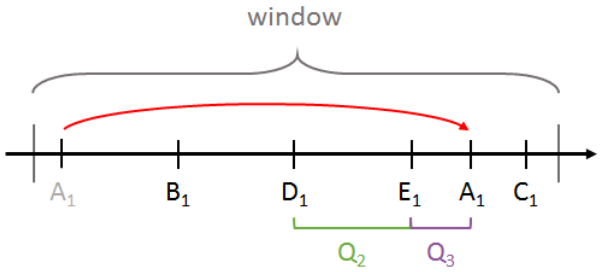


Figure 4.6: Modified event stream with delayed event instance A_1. Queries Q_2 and Q_3 have a match. The reordering algorithm introduced a false-positive because of Q_3.

In this example, it is also possible to reorder the event stream without a false-positive or false-negative just by taking the second ordering relation $or(E, C)$ of the private pattern. With negation of $or(E, C)$ the modified event stream is shown in Figure 4.7. This event stream has only a match of Q_2. Therefore, the reordering approach introduces neither a false-positive nor a false-negative but it successfully conceals the private pattern.

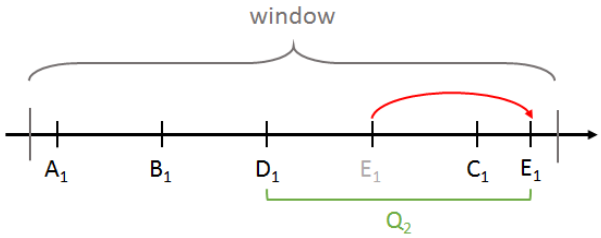


Figure 4.7: Modified event stream with delayed event E_1. The private pattern is concealed without introducing a false-positive or false-negative.

The examples above clearly show that the chosen events for reordering have a huge impact on the number of false-positives and false-negatives. A possible way to conceal a private pattern is to negate one of its ordering relations. The events which participate in that specific ordering relation are the events which should be reordered. Due to the fact that a private pattern often consists of multiple ordering relations, the reordering approach needs a mechanism to choose an ordering relation for negation out of the private pattern, which leads to a minimum number of false-positives and false-negatives. The quality of the result is not only influenced by the chosen events for reordering, also the position where these events are placed on has an impact. The next section describes this impact in detail.

4.1.2 Impact of Potential Positions for Reordering

The second problem is to find the positions where the events for reordering should be placed on. In the examples of the previous section 4.1.1, the events for reordering are just placed on one possible position. But in most cases, there are several positions with a different impact on the result in terms of number of false-positives and false-negatives. Once again, consider the example of section 4.1, to recap, this are the queries and private pattern:

$Q_1 = (B, C, D)$ $\qquad$ $P_1 = (A, C, E)$

$Q_2 = (D, E)$

$Q_3 = (E, A)$

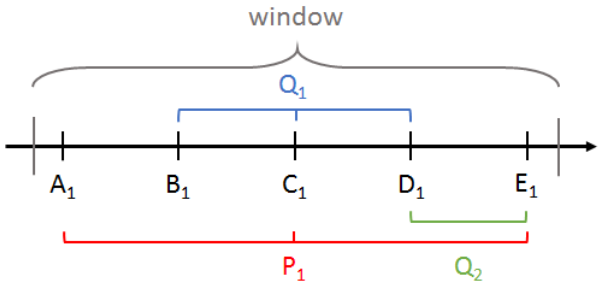


Figure 4.8: Input event stream with a match of Q_1, Q_2 and private pattern P_1.

Figure 4.8 shows the window with the event stream. The private pattern P_1 and the queries Q_1 and Q_2 have a match. Assume the reordering approach takes the ordering relation $or(A, C)$ for negation. To negate $or(A, C)$ it is either possible to delay event A to a position after event C or move C to a position before A. Figure 4.9 shows the case where event C is moved to a position before A. As expected, the private pattern does not match. The query Q_2 still has a match, but Q_1 has no match anymore. These positions for the reordered events introduces one false-negative.

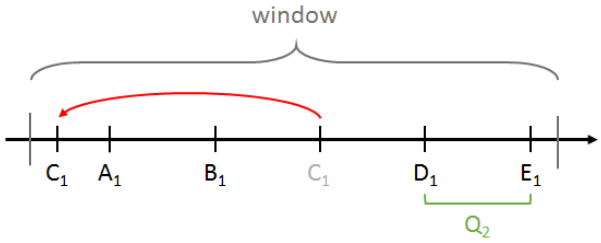
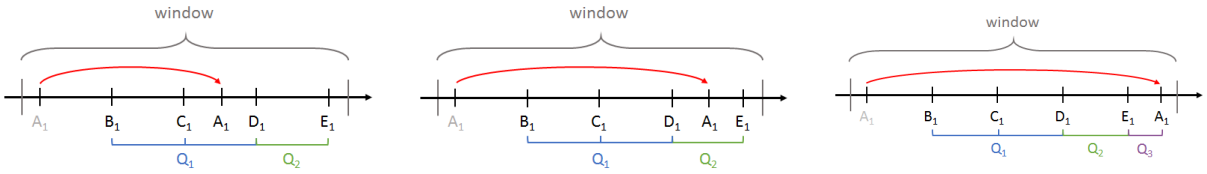


Figure 4.9: Modified event stream with moved event instance C_1. A false-negative is introduced because Q_1 has no match anymore.

Therefore, it might be better to negate the ordering relation $or(A, C)$ by delaying event A to a position after event C. There are several possible positions to place event A. Figure 4.10 shows the three possible modified event streams. All event streams contain no match for the private pattern P_1. Furthermore, each of them has a match for query Q_1 and Q_2. Therefore, none of the event streams introduces a false-negative. But the third event stream (Figure 4.10c) has an additional match for query Q_3. Thus, the third possible solution is not preferable because it adds a false-positive.

The last two sections show the huge number of possible solutions for the reordering approach. But not every solution achieves the same quality in terms of false-positives and false-negatives. Table 4.1 summarizes the possible solutions from the example of Section 4.1. One row represents one possible solution with the modified event stream, the chosen ordering relation for negation and the number of introduced false-positives and



(a) Match for Q_1 and Q_2. No false-negative or false-positive.

(b) Match for Q_1 and Q_2. No false-negative or false-positive.

(c) Match for Q_1, Q_2 and Q_3. Adds a false-positive because of Q_3.

Figure 4.10: Three possible modified event streams to negate the ordering relation $or(A, C)$ by delaying the event instance A_1.

Table 4.1: Summery of the possible solutions of the reordering approach to the example of section 4.1. The table considers the two different ordering relations of the private pattern P_1 which define the events for reordering. Additionally, the table shows the different positions where these events can be placed on.

Modified Event Stream	Negated Ordering Relation	Number of false-positives	Number of false-negatives
$C_1 \rightarrow A_1 \rightarrow B_1 \rightarrow D_1 \rightarrow E_1$	$or(A, C)$	0	1
$B_1 \rightarrow C_1 \rightarrow A_1 \rightarrow D_1 \rightarrow E_1$	$or(A, C)$	0	0
$B_1 \rightarrow C_1 \rightarrow D_1 \rightarrow A_1 \rightarrow E_1$	$or(A, C)$	0	0
$B_1 \rightarrow C_1 \rightarrow D_1 \rightarrow E_1 \rightarrow A_1$	$or(A, C)$	1	0
$A_1 \rightarrow B_1 \rightarrow E_1 \rightarrow C_1 \rightarrow D_1$	$or(C, E)$	0	1
$A_1 \rightarrow E_1 \rightarrow B_1 \rightarrow C_1 \rightarrow D_1$	$or(C, E)$	0	1
$E_1 \rightarrow A_1 \rightarrow B_1 \rightarrow C_1 \rightarrow D_1$	$or(C, E)$	1	1
$A_1 \rightarrow B_1 \rightarrow D_1 \rightarrow E_1 \rightarrow C_1$	$or(C, E)$	0	1

false-negatives. Every solution with the negation of ordering relation $or(C, E)$ introduces a false-negative. Expression 4.1 of the previous section 4.1.1 showed the reason that due to transitivity the ordering relation $or(C, E)$ is also needed by the matching queries and therefore, the negation of $or(C, E)$ definitely leads to a false-negative. A better choice is to negate the ordering relation $or(A, C)$. In this example it is possible to achieve a solution without any false-positives or false-negatives, with a good decision of the positions to place the events. Row 2 and 3 show these solutions.

But it also should be clear that the event reordering approach as well as the event suppression approach [WHRN13] cannot always achieve a solution without a false-positive and a false-negative. Section 3.2 already showed an example for the event suppression

approach in which all event types of the private pattern are used by matching queries and therefore, the suppression of one of these events leads to a false-negative. The following example illustrates this problem for the event reordering approach. Consider the same set of event types like in the examples before. The queries and the private pattern have this form:

$Q_1 = (A, B, C)$ $P_1 = (A, C, E)$

$Q_2 = (C, D)$

$Q_3 = (D, E)$

The window and the event stream is shown in Figure 4.11. All queries and the private pattern have a match. Thus, the reordering approach takes one of the ordering relations of the private pattern P_1 for negation, but both ordering relations, $or(A, C)$ and $or(C, E)$, are used by matching queries. The transitivity defines that $or(A, C)$ is used by query Q_1 and also that $or(C, E)$ is used by the combination of Q_2 and Q_3.

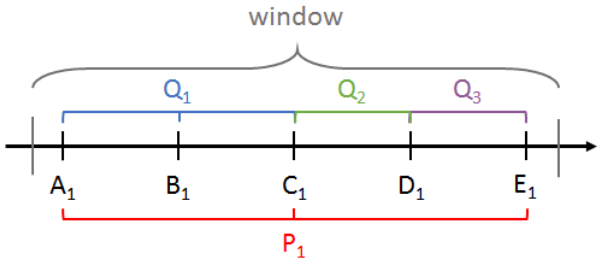
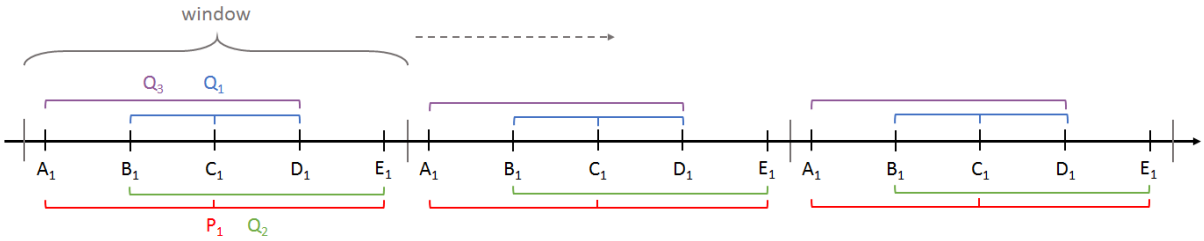


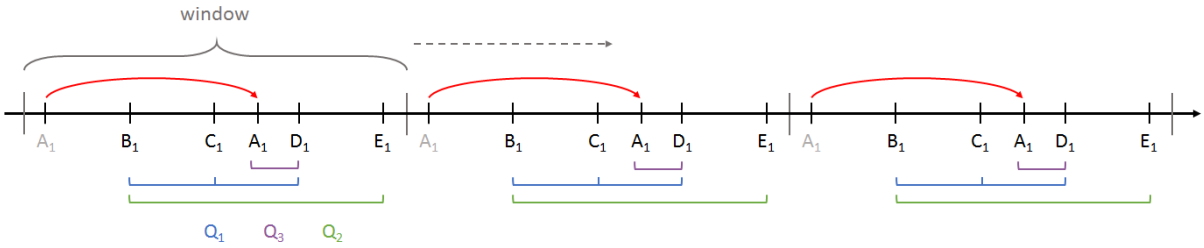
Figure 4.11: Input event stream which contains a match of Q_1, Q_2, Q_3 and P_1. The private pattern cannot be concealed without introducing a false-negative.

4.1.3 Impact of the Inter Arrival Rate

The inter arrival rate defines the time span from the arrival of one specific event type until the next occurrence of this event type at the system [Raq12]. The inter arrival rate does not directly influence the quality of the result of the reordering algorithm. The reordering algorithm rather has to consider the different types of inter arrival rates. In this thesis, we differentiate between three different types of inter arrival rates a system can have. First, the *same static inter arrival rates* where all event types are generated with the same static inter arrival rate. Second, the type of *different static inter arrival rates* where the different event types have different but static inter arrival rates and third, the *dynamic inter arrival rates* where the event generation of event types follow a certain probability distribution.



(a) Input event stream with events which have the same static inter arrival rate of 5.



(b) Modified event stream. Reordering only needs to be calculated for one window.

Figure 4.12: Input event stream (a) and the corresponding modified event stream (b) after reordering.

The most simple and unrealistic case is the type of same static inter arrival rates. All available sensors generate events with the same static inter arrival rate. If additionally the window has an equal size to the inter arrival rate, then every window contains the same set of events and the same order of these events. For instance, consider the same set of event types $\Sigma = \{A, B, C, D, E\}$ as in the previous section. These are the given queries and private pattern:

$$Q_1 = (B, C, D) \qquad P_1 = (A, C, E)$$

$$Q_2 = (B, E)$$

$$Q_3 = (A, D)$$

All event types have the same static inter arrival rate of 5. Then an event stream can have the form shown in Figure 4.12a. A window contains 5 events. All queries as well as the private pattern P_1 have a match in every window. In such a system the reordering algorithm only needs to solve the reordering step for one window. Afterwards, this solution can be used for every following window. A possible modified event stream of the reordering approach can have the form of Figure 4.12b. The modified event stream preserves all matches of the queries and conceals the private pattern. This is the most easiest case.

A more general assumption is that the window size differs from the inter arrival rate. Consider, for example, a window size of 4. Then the input event stream is splitted in windows as shown in Figure 4.13. Windows with same content will repeat periodically. Therefore, the reordering algorithm has to calculate the solution only for the number of windows of one cycle. For every following cycle, it is possible to use the already calculated solution.

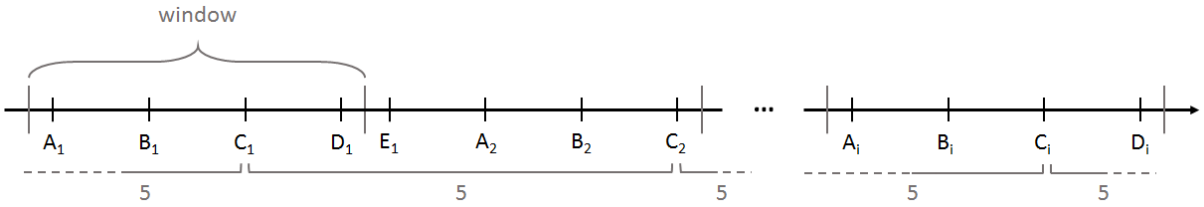


Figure 4.13: Window size is smaller than the static inter arrival rate of 5.

The different static inter arrival rates are closely related to the last example. Also this case is not very realistic. Difference to the same static inter arrival rates is that the event types can have different inter arrival rates. For instance, the event types have these different inter arrival rates: $(A; 10)$, $(B; 8)$, $(C; 5)$, $(D; 5)$, $(E; 7)$. Figure 4.14 shows one possible input event stream. Exactly as in the previous example, the windows with the same set of events and the same order between these events repeat periodically. After the calculation of the solution for one cycle the solution can also be applied for the following cycles.

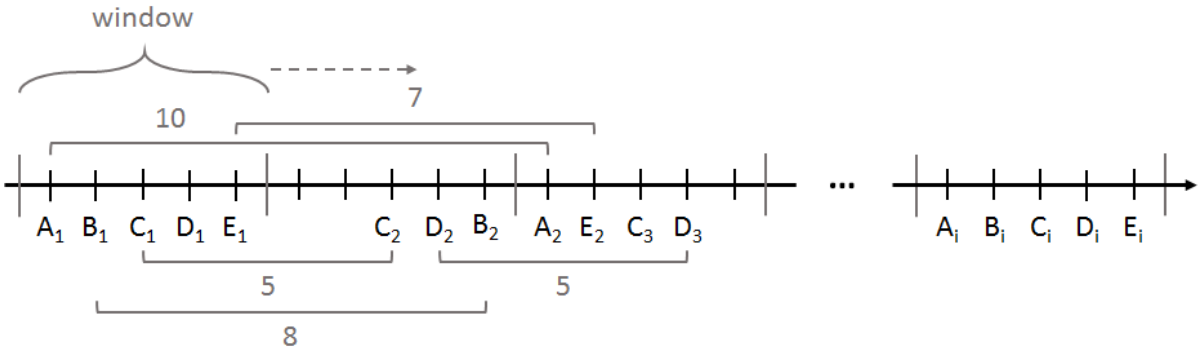


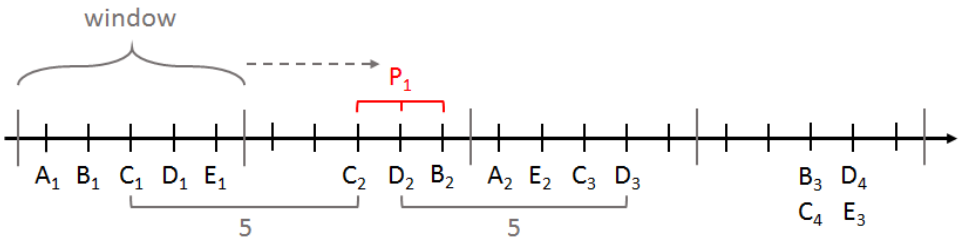
Figure 4.14: Input event stream with events which have different static inter arrival rates.

The most general case is the type of dynamic inter arrival rates. It is also the most realistic type of inter arrival rates for real world scenarios. The events are generated according to a certain probability distribution. For event generation, this thesis uses the *normal distribution*, also called *Gaussian distribution* [Weib]. This distribution takes

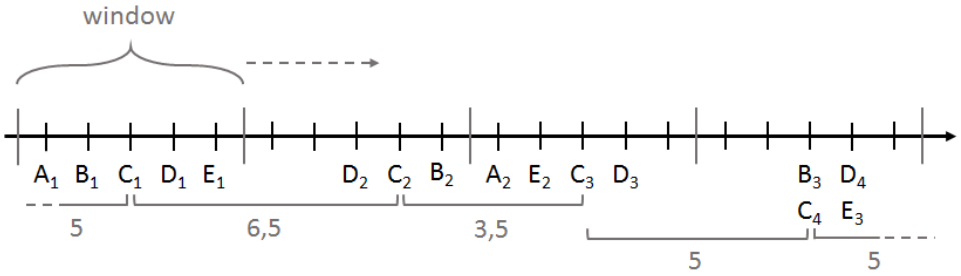
fluctuations of the inter arrival rates into account. The reordering algorithm has to calculate a solution for every window. In contrast to the other types, periodicity cannot be assumed in this type of inter arrival rates. This thesis focuses on the type of dynamic inter arrival rates because it is the most realistic one and the types of same static inter arrival rates and different static inter arrival rates can nearly be simulated with corresponding values for the mean [Weia] and the standard deviation [Weic] of the normal distribution.

4.1.4 ε-Range

Up to this point the reordering approach mainly focused on avoiding private pattern matches while trying to minimize the number of false-positives and false-negatives. Another goal is that an attacker cannot reveal the changes in the modified event stream. As soon as the reordering algorithm changes the position of one event instance, it also changes the inter arrival rate of this event type. Therefore, the reordering algorithm is restricted to reorder an event instance only in its specific ε-Range. The idea behind is to minimize the fluctuations of the inter arrival rates.

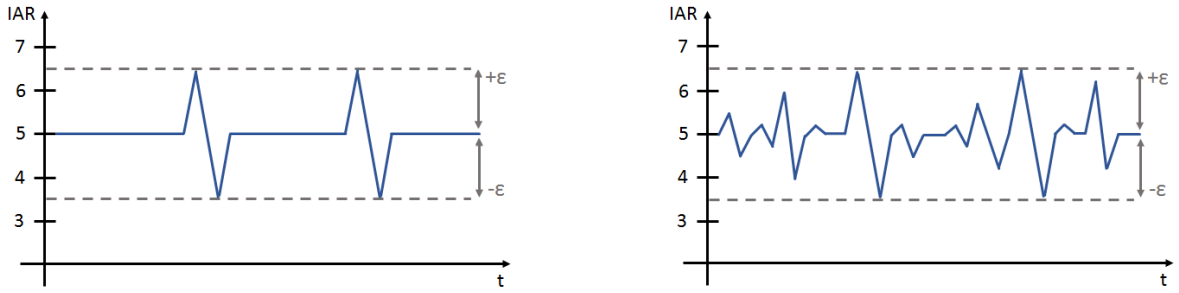


(a) Input event stream with events which have different static inter arrival rates.



(b) Modified event stream with changed inter arrival rate of event C.

Figure 4.15: Input event stream (a) and the corresponding modified event stream (b) with changed inter arrival rate.



(a) Event type with static inter arrival rate.

(b) Event type generated according to a probability distribution.

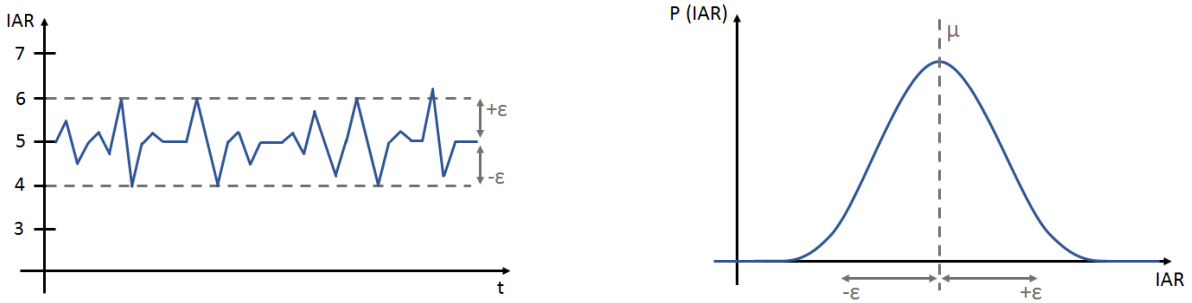
Figure 4.16: Monitored inter arrival rates.

The following example shows the problem of the changing inter arrival rates in the modified event stream. This is the set of event types $\Sigma = \{A, B, C, D, E\}$. For simplicity, this example uses different static inter arrival rates: $(A; 10)$, $(B; 8)$, $(C; 5)$, $(D; 5)$, $(E; 7)$. Also no queries and only the following private pattern is considered:

$$P_1 = (C, D, B)$$

The given input event stream is shown in Figure 4.15a. The private pattern P_1 has a match in the second window. Thus, the reordering algorithm will change the order in that window to conceal the match of P_1. One possibility is to delay event instance C_2 to a position after event instance D_2. Figure 4.15b shows the possible modified event stream. As expected, the modified event stream contains no match of the private pattern P_1. In this example, we are not interested in false-positives and false-negatives but in the changed inter arrival rate of event type C. The inter arrival rate from C_1 to C_2 as well as from C_2 to C_3 is effected because of the changed position of C_2. Between C_1 and C_2 the inter arrival rate increases from value 5 to $6,5$ and between C_2 and C_3 it decreases from 5 to $3,5$. Before event instance C_1 and after C_3 the inter arrival rate stays constant at the defined value of 5 as long as no private pattern match occurs.

The problem with these changes of the inter arrival rate is that an attacker might detect these anomalies if they occur very rarely. If an attacker monitors the modified event stream with the events and the corresponding inter arrival rates, it could be possible to create a chart like in Figure 4.16a. The chart shows the monitored inter arrival rate of one specific event type over a period of time. Most of the time the inter arrival rate stays constant at a value of 5, but there are two exceptions where the inter arrival rate first increases to a value of $6,5$, then decreases to $3,5$ and afterwards goes back to 5. Such a chart could be the monitored result of the previous example. The changes of the inter arrival rate are easy to see. With that knowledge it could be possible to restore the input event stream. But as already mentioned, the type of different static inter arrival rates,



(a) ε-Range smaller than some fluctuations.

(b) Normal distribution.

Figure 4.17: Monitored inter arrival rate (a) and the normal distribution (b) according to which the event type is generated.

which is used in the example above and in the chart in Figure 4.16a, is very unrealistic for the real world. In real world scenarios fluctuations are in the inter arrival rates because there are many factors, for example the network, which influence the inter arrival rate. This is why we focus on the type of dynamic inter arrival rates, which is a more realistic type. Figure 4.16b shows the same chart as in Figure 4.16a but with dynamic inter arrival rates. The difference is that the inter arrival rate does not stay at a constant value of 5, but fluctuates around this value because of the standard deviation of the normal distribution. Therefore, the two anomalies with the changed inter arrival rates are a lot harder to see.

Both Figures 4.16a and 4.16b show the defined ε-Range. The ε-Range defines the maximum range an event can be delayed or moved forward on the event stream. According to the drawn ε-Ranges in the charts the reordering algorithm uses the full range for concealing the private pattern in the example above. Goal of the ε-Range is that an attacker cannot recognize any anomalies as it is possible in Figure 4.16a and 4.16b. Figure 4.17a shows a chart with an ε-Range where the changes of the inter arrival rate are not recognizable because the ε-Range is smaller than the fluctuating inter arrival rate values. Therefore, the ε-Range heavily depends on the normal distribution, according the event type is generated. Figure 4.17b shows such a normal distribution. In this thesis, the *mean* μ [Weia] represents the defined inter arrival rate for this event type and the *standard deviation* σ [Weic] and the *variance* σ^2 [Weid] defines the size of the possible fluctuations. The ε-Range should be smaller than the possible size of the fluctuations because extreme values are unusual according to the probability distribution and the reordering approach might introduce a lot of them. Therefore, an ε-Range which is smaller than the fluctuating inter arrival rate values is not enough to conceal all the modifications. Additionally, also the probability distribution, according to which an event type is generated, should not change because of the modification of the event stream.

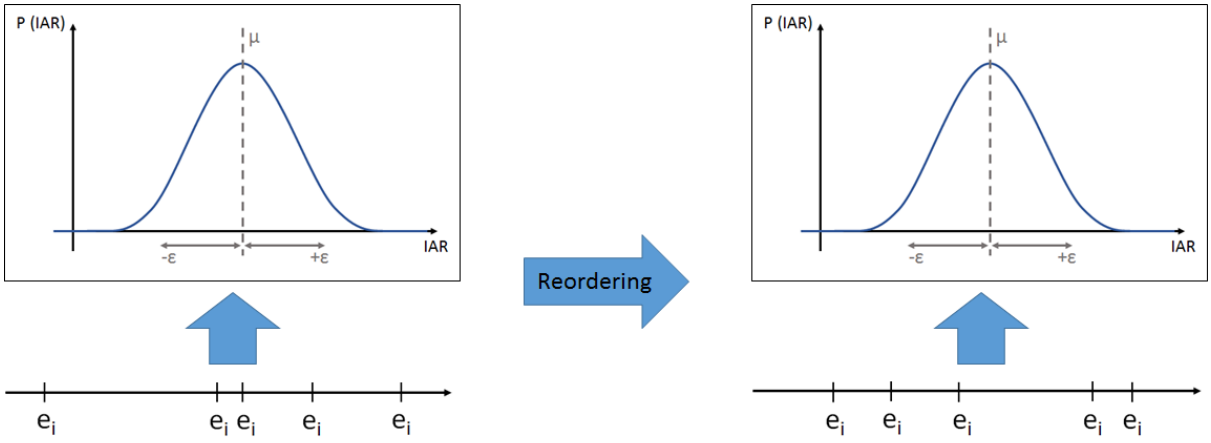


Figure 4.18: The probability distribution of an event type of the monitored event stream should be the same as of the original event stream.

Figure 4.18 shows this scenario. The probability distribution of an event type of the monitored event stream should be the same as of the original event stream. This thesis only evaluates the impact of different sizes of the ε-Range on the reordering algorithm because the reordering algorithm cannot always find a solution without a private pattern match if the restriction of the ε-Range is too hard. Therefore, there is a tradeoff between concealing all private patterns of the event stream and hiding the inter arrival rate changes in the modified event stream. Achieving the condition that the probability distribution does not change because of the modification, is an interesting area for future work and is not investigated in this thesis.

4.2 Graph-based Reordering Algorithm

This thesis presents two event stream reordering algorithms. This section gives a detailed description of the graph-based algorithm. The matching queries and private patterns of the corresponding window build a directed acyclic graph (DAG). The graph $G = (E, R)$ consists of a set of event instances $e \in E$ and a set of ordering relations $or(e_i, e_j) \in R \subseteq (E \times E)$. The algorithm transforms this graph to produce a modified event stream without private pattern matches.

4.2.1 Graph Generation

The matching queries and private patterns of one window build a DAG. All event instances which participate in a matching query represent a node of the graph. The ordering relations of the matching queries define the edges of the graph. Additionally, the event instances of the ordering relation for negation of the private pattern represent two nodes and the negated ordering relation represent the corresponding edge. For instance, consider the same set of event types like in the previous sections. These queries and the private pattern are given:

$Q_1 = (B, C, D)$ $\qquad$ $P_1 = (A, C, E)$

$Q_2 = (B, E)$

$Q_3 = (A, D)$

The window consists of the event stream with annotated arrival times shown in Figure 4.19. All queries and the private pattern have a match in this window. We start with Q_1 for the graph generation. First of all, three nodes, B, C and D, are added to the graph. For the two ordering relations $or(B, C)$ and $or(C, D)$ of Q_1 the graph gets the edges (B, C) and (C, D). Figure 4.20a shows the current graph. After adding the nodes and

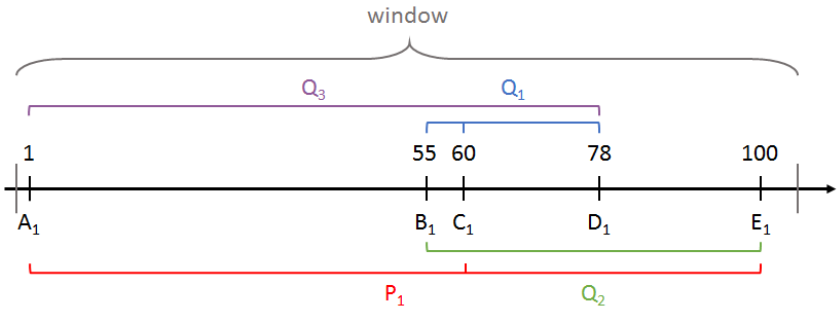
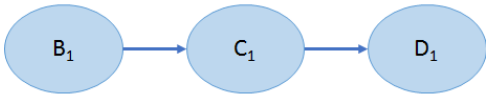
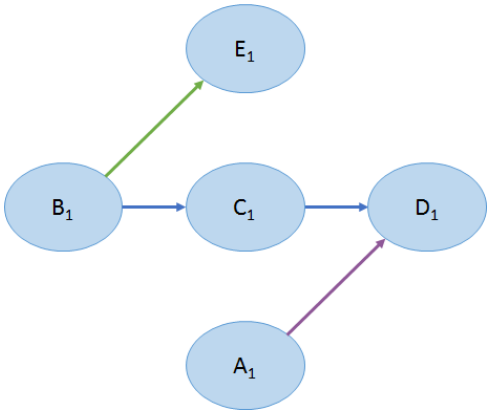


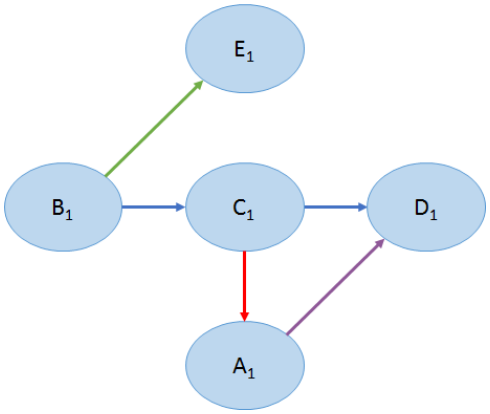
Figure 4.19: Input event stream with annotated inter arrival times.



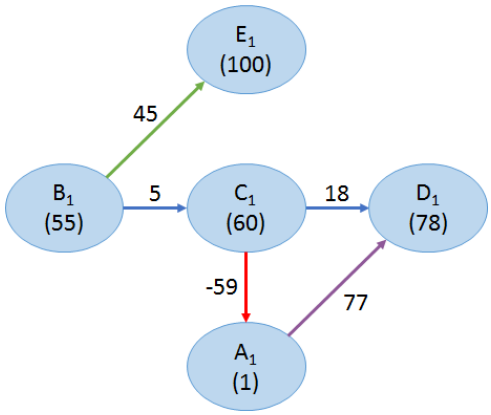
(a) Graph consists of Q_1.



(b) Graph consists of all queries.



(c) Graph consists of all queries and the ordering relation for negation.



(d) Full graph with annotated arrival times.

Figure 4.20: Graph generation steps.

edges for query Q_2 and Q_3, the graph has the form shown in Figure 4.20b. Next step is to consider the private pattern and the chosen ordering relation for negation. How to choose the ordering relation for negation is discussed in Section 4.2.4. Assume the ordering relation for negation of the private pattern is $or(A, C)$. Therefore, the private pattern would add node A and C if they are not already part of the graph. The negated ordering relation $or(A, C)$ adds the edge (C, A). Note that this edge is directed from C to A. Figure 4.20c shows the resulting graph.

The graph of Figure 4.20c contains all ordering relations of the queries. If all ordering relations hold, then the queries have a match. Furthermore, the graph contains a negated ordering relation of the private pattern. If this ordering relation holds, then the private pattern has no match. Thus, the goal of the reordering algorithm is to generate an event stream which fulfills all ordering relations of the graph.

Algorithm 4.1 Topological Sorting [Kah62]

```
procedure TOPOLOGICALSORTING(g)
    sortedList ← [ ]
    while ISNOTEMPTY(g) do
        n ← GETNODEWITHOUTINCOMMINGEDGES(g)
        sortedList ← n
        REMOVEALLOUTGOINGEDGES(n)
        REMOVENODE(g,n)
    end while
end procedure
```

Up to this point, no arrival times are taken into account. The given input event stream shows the different arrival times of the event instances. Therefore, every node additionally gets labeled with the arrival time of the corresponding event instance. The arrival time of event instance e is denoted by $t(e)$. For instance, the node representing event instance C_1 has the label $C_1(60)$. Finally, all edges get a label of its edge weight.

(4.2) $w(e_i, e_j) = t(e_j) - t(e_i)$

Equation 4.2 defines the weight w of edge (e_i, e_j). The edge weight is equal to the arrival time difference between the source and target event instance of the edge. For instance, edge (C_1, D_1) has the weight $w = 78 - 60 = 18$. The complete graph with annotated nodes and edges is shown in Figure 4.20d.

4.2.2 Algorithm

Input for the reordering algorithm is the generated graph. At the beginning, for simplicity the algorithm does not look at arrival times and the ε-Range. Therefore, the considered input graph is shown in Figure 4.20c. As already mentioned, goal is to fulfill all ordering relations of the given graph. In other words, the algorithm is looking for a linear order of the nodes such that all edges (ordering relations) are fulfilled in that linear order.

Topological sorting [Kah62] is exactly doing this. Algorithm 4.1 shows a simple example code for topological sorting. The input to the procedure is a graph g. If the graph is a directed acyclic graph, this code returns the topological sorting in *sortedList*. As long as the graph contains a node, it takes one node n without any incoming edges. At the beginning, the graph must have a node without an incoming edge, otherwise the graph is not acyclic. Then the algorithm adds this node n to the topological sorted list, removes all outgoing edges of n and removes n from the graph.

Algorithm 4.2 Graph Transformation

procedure GRAPHTRANSFORMATION(g)
  while CONTAINSEDGEWITHNEGATIVEORZEROWEIGHT(g) **do**
    $edge \leftarrow$ GETEDGEWITHNEGATIVEORZEROWEIGHT(g)
    $sNode \leftarrow$ GETSOURCENODE($edge$)
    $tNode \leftarrow$ GETTARGETNODE($edge$)
    $sNode.Time \leftarrow sNode.Time - \lfloor \frac{|edge.weight|}{2} \rfloor$
    $tNode.Time \leftarrow tNode.Time + \lceil \frac{|edge.weight|}{2} \rceil + 1$
    UPDATEEDGEWEIGHTS(g)
  end while
end procedure

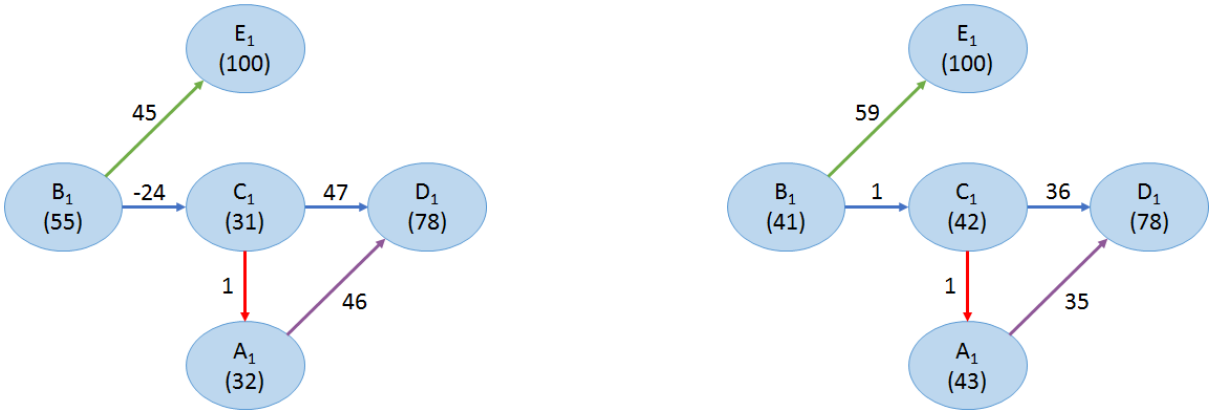
One possible topological sorting of the graph in Figure 4.20c can look like this:

B, C, A, D, E

Assume this order for the modified event stream. All matching queries of the example in the previous section 4.2.1 still match and the private pattern is concealed. But topological sorting cannot care about arrival times or the ε-Range. Therefore, another algorithm is needed.

From now on, the reordering approach considers the graph of Figure 4.20d and cares about the arrival times. All ordering relations of the graph have to be fulfilled. We know that the ordering relations of the matching queries are already fulfilled, otherwise they would not have a match. The only ordering relation which is not fulfilled is the negated $or(A, C)$ of the private pattern. Edge (C_1, A_1) of the graph represents the negated ordering relation. Note that this edge is the only edge with a negative weight. The weight defines the arrival time difference of the connected nodes which represent the event instances. Therefore, a negative weight means that the currently used arrival times do not fulfill the desired ordering relations. Goal of the algorithm is, to transform the graph in such a way that no edge with a negative weight exists.

Algorithm 4.2 shows the graph transformation code. As input the procedure gets a directed acyclic graph like the one in Figure 4.20d. As long as the graph contains an edge with negative or 0 weight, it does the following: The algorithm takes such an edge with negative or 0 weight. Then it gets the corresponding source and target node of that edge. In the next step, the algorithm updates the arrival time of the source and target node. Both, the arrival time of the source as well as of the target node, get modified by nearly the same value, which is the half of the edge weight. Result of this modification is a positive edge weight at least with a value of 1. At the end, the procedure updates changing edge weights of the graph.



(a) Graph after first while loop.

(b) Resulting graph of the graph transformation.

Figure 4.21: Changes of the graph transformation algorithm.

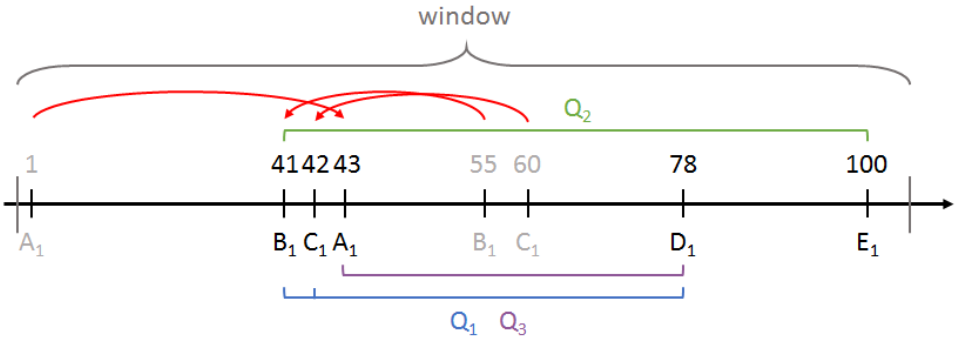


Figure 4.22: Resulting event stream of the graph-based reordering algorithm.

The graph of Figure 4.20d has only one negative edge (C_1, A_1). Therefore, the algorithm takes this edge for transformation. The modified arrival time of the source node is calculated by $60 - \lfloor \frac{|-59|}{2} \rfloor = 31$ and for the target node by $1 + \lceil \frac{|-59|}{2} \rceil + 1 = 32$. According to equation 4.2, the new edge weight is $32 - 31 = 1$. Because of the modified arrival times, also other edge weights of the graph need to be updated. The resulting graph after the first while loop is shown in Figure 4.21a. Figure 4.21b shows the end result of the graph transformation. As expected, the graph contains no edge with a negative weight. The resulting arrival times of the nodes define the exact position of the corresponding event instance in the modified event stream. Therefore, the modified event stream is shown in Figure 4.22. All queries have a match in the event stream. The private pattern is concealed and neither false-positives nor false-negatives are introduced. But the algorithm still does not take the restriction of the ε-Range into account. For this reason, we developed the reordering condition which is discussed in the next section.

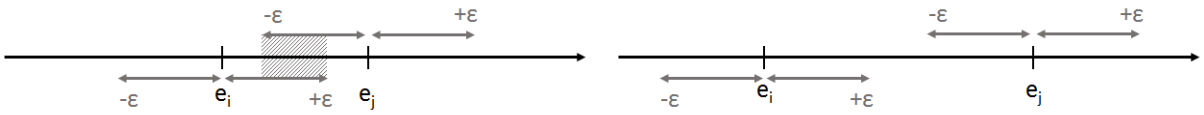
4.2.3 Reordering Condition

The idea of the *reordering condition* is to check whether it is possible to reorder an event stream under restriction of the ε-Range or not. Therefore, the reordering condition has to check every possible path $P(n_i, n_j)$ with source node n_i and target node n_j in the generated graph G. Actually it is enough to check every path which contains at least one edge with a negative weight because only those paths can contradict the reordering condition. The reordering condition is defined as follows:

$$\forall\ P(n_i, n_j) \in G : \varepsilon(n_i) + \varepsilon(n_j) \geq -\sum_{e \in P} e.weight + |P| \tag{4.3}$$

The sum over the edge weights of a path define the arrival time difference of the source and target node. The nodes are in the right order if the sum is positive, but if the sum is negative, then the order of the source and target node is not the right one for a solution. Thus, the reordering algorithm has to change the order of these two nodes which represent the event instances. The reordering is only possible if the sum of the edge weights plus the number of edges in the path is smaller or equal to the sum of the ε-values of the source and target node. The number of edges is considered because the resulting arrival times should differ at least by a value of 1 to fulfill the needed ordering relation.

Figure 4.23 shows the reordering condition visually. The reordering condition checks for two event instances, which have to be reordered, whether their ε-Ranges overlap or not. The reordering is possible if the ε-Ranges overlap (Figure 4.23a), otherwise reordering of these event instances is not possible (Figure 4.23b).



(a) Reordering condition fulfilled. **(b)** Reordering condition not fulfilled.

Figure 4.23: Visually interpretation of the reordering condition.

4.2.4 Choosing an Ordering Relation of the Private Pattern

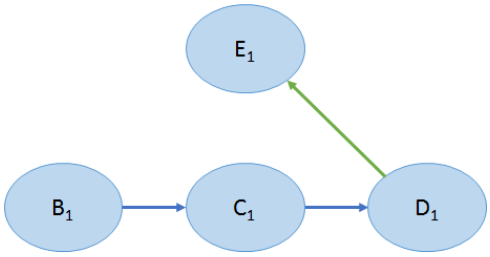
One unsolved problem is still to choose a good ordering relation for negation of the private pattern. Section 4.1.1 already showed that this decision has an impact on the number of false-positives and false-negatives. This section considers the same example as Section 4.1.1. To recap, this is the set of event types $\Sigma = \{A, B, C, D, E\}$. The queries and private pattern have the following form:

$Q_1 = (B, C, D)$ $\quad$ $P_1 = (A, C, E)$

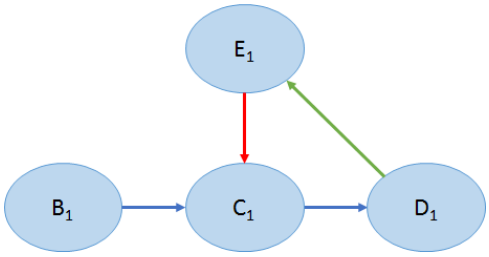
$Q_2 = (D, E)$

$Q_3 = (E, A)$

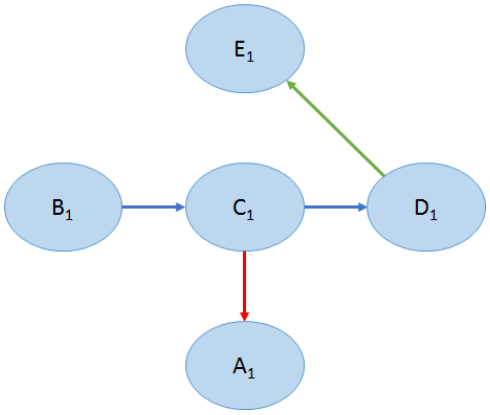
The input event stream is the same as in Figure 4.1. Therefore, query Q_1 and Q_2 and the private pattern have a match. The generated graph out of the matching queries is shown in Figure 4.24a. Now, the algorithm has to choose an ordering relation for negation of the private pattern to add the corresponding edge to the graph.



(a) Graph consists of Q_1 and Q_2.



(b) Graph with cycle C_1, D_1, E_1, C_1.



(c) Resulting event stream of the graph-based reordering algorithm.

Figure 4.24: Graph generation with the ordering relation for negation.

First of all, the algorithm gets all possible ordering relations for negation. Private pattern P_1 has to possible ordering relations, $or(A, C)$ and $or(C, E)$. Then, it sorts the ordering relations according to the absolute arrival time distance $d(e_i, e_j) = |e_i.time - e_j.time|$ of the participating event instances. $or(A, C)$ has a distance of $d(A, C) = |1 - 60| = 59$ and $or(C, E)$ of $d(C, E) = |60 - 100| = 40$. With a lower distance it is more likely that the reordering condition can be fulfilled. The algorithm would take ordering relation $or(C, E)$ for negation because of the lower distance. Thus, the algorithm adds edge (E, C) into the graph. Figure 4.24b shows the resulting graph. Section 4.1.1 already discussed that $or(C, E)$ is a bad choice and will introduce a false-negative. The expression 4.1 showed due to transitivity that the ordering relation is also used by the matching queries. This effect can also be seen in the graph because the adding of edge (E, C) introduces a cycle (C_1, D_1, E_1, C_1). Therefore, the algorithm deletes edge (E, C) and takes the other ordering relation $or(A, C)$ of private pattern P_1. The graph with added edge (C, A) is shown in Figure 4.24c. This graph has no cycle, so reordering does not introduce a false-negative. But also the reordering condition must hold, otherwise reordering is not possible because of the ε-Range. If the reordering condition holds, the reordering algorithm will get a solution without a private pattern and without a false-positive or false-negative. But assume that the reordering condition does not hold. Then the algorithm takes the other ordering relation $or(C, E)$ and deletes one edge out of the cycle. It chooses the edge which is used by the least number of queries. Thus, the algorithm also introduces only the least number of false-negatives. If no ordering relation of the private pattern fulfills the reordering condition, the algorithm cannot generate a modified event stream without the private pattern match because of the ε-Range.

If the reordering of an event stream is not possible without a false-negative, the algorithm introduces the least possible number of false-negatives. But up to this point, no mechanism takes the number of false-positives into account. Therefore, the next Section 4.2.5 presents the strategy to avoid the introduction of false-positives.

4.2.5 Avoidance of False-Positives

Amongst other things, the last Section 4.2.4 described how the algorithm avoids the introduction of false-negatives. But the algorithm also tries to avoid the introduction of false-positives. Consider the same example like in the last Section 4.2.4. The generated graph is the one in Figure 4.24c. The only query which does not have a match in the input event stream is query Q_3. In worst case, a possible modified event stream according to the generated graph can have the form shown in Figure 4.25.

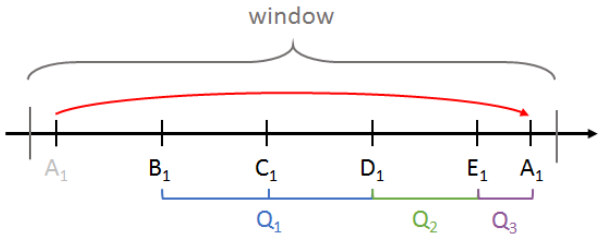


Figure 4.25: Possible modified event stream of the reordering algorithm.

This solution introduces a false-positive because the event stream contains a match for query Q_3. To prevent this, the algorithm handles a not matching query similar to a private pattern. It chooses one of the ordering relations of the query for negation. In this case, the query has only one ordering relation $or(E, A)$. Therefore, the algorithm adds the edge (A, E) to the graph, checks for a cycle and for the reordering condition. If no cycle is introduced and the reordering condition is not contradicted, it is guaranteed that the algorithm does not introduce a false-positive of this query. Otherwise, the reordering step is not possible without the introduction of a false-positive. Figure 4.26 shows the graph with edge (A, E). With this graph the modified event stream from above is not a possible result anymore.

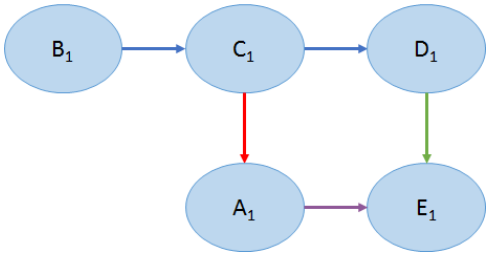


Figure 4.26: Graph with edge to avoid false-positive.

4.2.6 Multiple Occurrences of the Same Event Type

Up to this point, all examples had only one event instance of a certain type in the window. The reordering algorithm has to consider the case where multiple event instances of the same event type occur in a window because, for example, the ordering relation of the private pattern has to be negated for every event occurrence of the participating event types. The following example shows the issue. Assume the same set of event types, queries and the private pattern of Section 4.2.4. Figure 4.27 shows the input event stream.

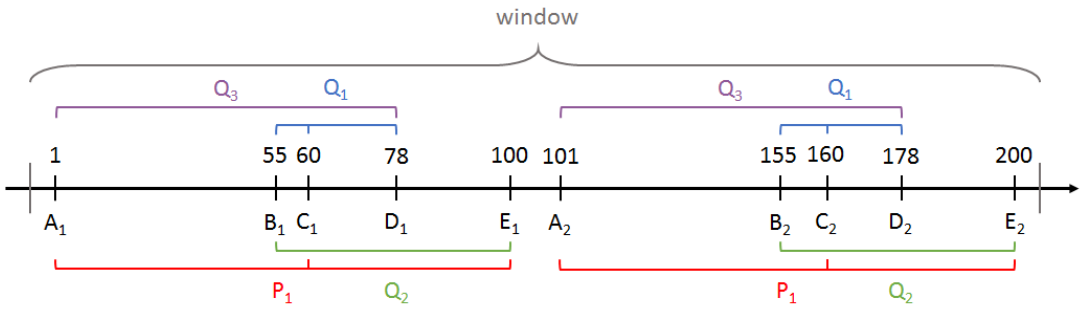
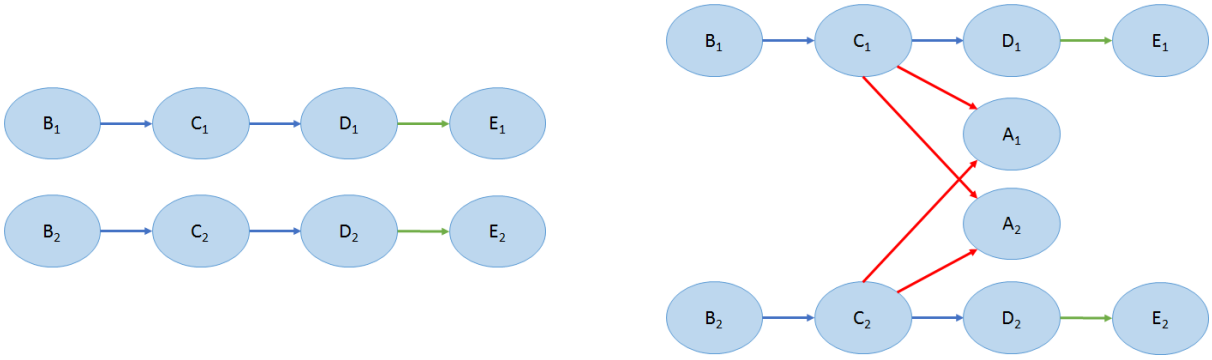


Figure 4.27: Input event stream with a double sized window.



(a) Graph consists of Q_1 and Q_2.

(b) Graph with all edges corresponding to the ordering relation for negation.

Figure 4.28: Graph generation with multiple events of the same type.

The window has double size and it contains the 10 given event instances. According to the used selection and consumption policy, the queries Q_1 and Q_2 have two matches as well as the private pattern P_1. The graph generation for the matches of the queries work exactly like already discussed. The graph in Figure 4.28a shows the generated graph. For the private pattern it differs a bit. First of all, the number of matches of a private pattern does not matter. Important is the number of event instances of an event type which participates in the ordering relation for negation of the private pattern. As we already know, the ordering relation $or(A, C)$ is a good choice for negation. Therefore, the algorithm would add edge (C, A) to the graph. But because of the multiple instances of an event type, the algorithm needs to add the edge (C, A) for every possible combination of these event instances. Thus, a total of 4 edges is added with the combinations (C_1, A_1), (C_1, A_2), (C_2, A_1) and (C_2, A_2). The complete graph is shown in Figure 4.28b. Same holds for the avoidance of false-positives. Rest of the algorithm works as already discussed.

4.2.7 Different Granularities

The graph-based reordering algorithm is able to use different granularities for the arrival time changes in the graph transformation Algorithm 4.2. The used granularity has an impact on the resulting arrival times of the reordered event instances. For instance, the used granularity in Algorithm 4.2 has the effect that the reordered event instances have arrival times which are very close to each other. This thesis considers the following three different granularities:

(4.4a)
$$\frac{|edge.weight|}{2}$$

(4.4b)
$$\varepsilon(n_i)$$

(4.4c)
$$\lceil \frac{\varepsilon(n_i)}{2^k} \rceil, k = 1, \ \ldots, \ number\ of\ iteration$$

The granularity of 4.4a is already used in Algorithm 4.2. It changes the inter arrival times of the event instances by the half of the *edge.weight* which is equivalent to the difference of the arrival times. As already mentioned, the effect is that the resulting arrival times lie very close to each other. The granularity of 4.4b changes the arrival time by the ε-value. Therefore, the full available range is always used for reordering. In contrast to granularity 4.4a, the resulting arrival times tend to have a big distance to each other. Granularity 4.4c is more an iterative way. With the increasing number of iterations, the arrival time changes of the event instances become more and more fine-grained.

4.3 ILP-based Reordering Algorithm

The second reordering algorithm presented in this thesis is based on an Integer Linear Programming (ILP) formulation. The objective function wants to minimize the total arrival time changes of the event instances. The matching queries and private patterns of a window as well as the ε-Range define the constraints of the ILP formulation.

4.3.1 ILP Formulation

Input for the ILP formulation is a set of event instances E of size n, a set of ordering relations R and a set of ordering relations N. Furthermore, the ILP formulation needs all $\varepsilon(e_i)$-values of the different event instances e_i to fulfill the ε-Range. $x(e_i)$ denotes the arrival time changes of an event instance. The event instances of the considered window are part of E. The set R consists of the ordering relations of the matching queries whereas N consists of the ordering relations for negation of the matching private patterns. N consists only of one ordering relation because this thesis considers only one private pattern. The ILP formulation is defined as follows:

$$
\begin{aligned}
&\text{minimize} && \sum_{e_i \in E} |x(e_i)| \\
&\text{subject to} && \forall\, or(e_j, e_k) \in R: && e_j.time + x(e_j) < e_k.time + x(e_k) \\
& && \forall\, or(e_j, e_k) \in N: && e_k.time + x(e_k) < e_j.time + x(e_j) \\
& && && -\varepsilon(e_i) \leq x(e_i) \leq \varepsilon(e_i) \qquad i = 1,\ \ldots,\ n
\end{aligned}
$$

The objective function minimizes the total arrival time changes of all event instances. Every ordering relation $or(e_j, e_k)$ of a matching query adds a constraint $e_j.time + x(e_j) < e_k.time + x(e_k)$ to the ILP. The constraint specifies that the changed arrival time of event instance e_j has to be smaller than the changed arrival time of e_k. If this is the case, the ordering relation is fulfilled. The ordering relation $or(e_j, e_k)$ for negation of the private pattern adds an constraint $e_k.time + x(e_k) < e_j.time + x(e_j)$ which is defined the other way around as the one for the queries. Therefore, the constraint defines that the changed arrival time of e_j has to be greater than the one of e_k because the ordering relation should be unfulfilled. Finally, the ILP formulation contains constraints for the restriction of the ε-Range. The constraints of the form $-\varepsilon(e_i) \leq x(e_i) \leq \varepsilon(e_i)$ assure that no arrival time will be changed with a greater value than the allowed ε-Range of this event instance. The result of the ILP is the difference $x(e_i)$ of the original arrival time for every event instance e_i. Then the algorithm only needs to place the event instances on the modified event stream according to the updated arrival times $e_i.time = e_i.time + x(e_i)$.

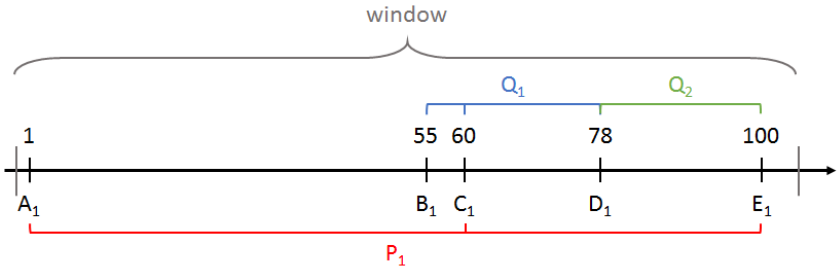


Figure 4.29: Input event stream.

The following example shows the generation of the ILP formulation. Consider the available set of events $\Sigma = \{A, B, C, D, E\}$. The queries and the private pattern have the following form:

$$Q_1 = (B, C, D) \qquad P_1 = (A, C, E)$$

$$Q_2 = (D, E)$$

$$Q_3 = (E, A)$$

The window and the input event stream is shown in Figure 4.29. The private pattern P_1 and the queries Q_1 and Q_2 have a match. Therefore, set R contains these ordering relations $R = \{or(B_1, C_1), or(C_1, D_1), or(D_1, E_1)\}$. Assume ordering relation $or(A, C)$ is taken for negation, then set N has this form $N = \{or(A_1, C_1)\}$. In set E are all event instances of the window $E = \{A_1, B_1, C_1, D_1, E_1\}$. The example uses $\varepsilon(e_i) = 50$. The resulting ILP formulation looks as follows:

$$\begin{aligned}
\text{minimize} \quad & \sum_{e_i \in E} |x(e_i)| \\
\text{subject to} \quad & 55 + x(B_1) < 60 + x(C_1) \\
& 60 + x(C_1) < 78 + x(D_1) \\
& 78 + x(D_1) < 100 + x(E_1) \\
& 60 + x(C_1) < 1 + x(A_1) \\
& -50 \leq x(e_i) \leq 50 \qquad i = 1, \ \ldots, \ n
\end{aligned}$$

The solution of the ILP returns these arrival time differences:

$$x(A_1) = 50; \ x(B_1) = -6; \ x(C_1) = -10; \ x(D_1) = 0; \ x(E_1) = 0$$

Figure 4.30 shows the resulting modified event stream. The stream has a match for Q_1 and Q_2. The private pattern is concealed and neither a false-positive nor a false-negative is introduced.

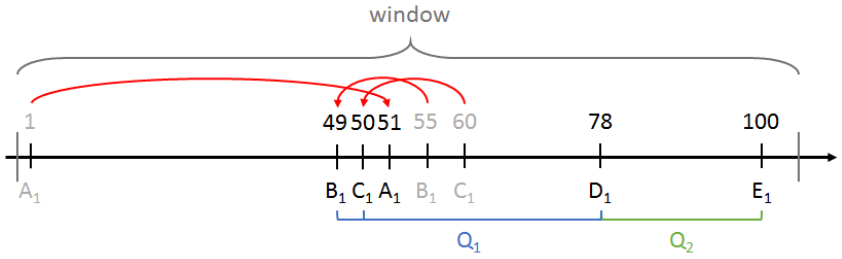


Figure 4.30: Modified event stream of the ILP-based algorithm.

4.3.2 Choosing an Ordering Relation

This algorithm has the same problem like the graph-based reordering algorithm. The ILP-based algorithm has to choose an ordering relation of the private pattern for negation. As we already know, the ordering relation should not be the same as one of the ordering relations of the matching queries and the restriction of the ε-Range needs to be fulfilled. But without checking the reordering condition, the algorithm never knows beforehand, whether the ILP can have a solution or not. Therefore, one possible way is to try all ordering relations of the private pattern in the ILP until a solution is found. The other way is to generate the graph of the graph-based algorithm and out of that get the ordering relation for negation.

4.3.3 Avoidance of False-Positives

The ILP-based algorithm has the possibility to avoid false-positives because a non-matching query can be treated the same way like a private pattern. Therefore, the algorithm only needs to choose one ordering relation for negation of the query and add the corresponding constraint to the ILP formulation in exact the same way as for a private pattern. If the ILP formulation has a solution, it is guaranteed that this query does not introduce a false-positive in the modified event stream. But as already mentioned in the last Section 4.3.2, the ILP-based algorithm has no opportunity to check beforehand, whether the ILP formulation with the taken ordering relation for negation has a solution or not. Thus, the algorithm has the same possibilities as in the previous Section 4.3.2, either to try all ordering relations of the non-matching query until the ILP has a solution or to generate the graph of the graph-based algorithm and choose the ordering relation according to the graph.

4.3.4 Multiple Occurrences of the Same Event Type

The ILP-based algorithm also has to consider the case that multiple event instances of the same event type occur in one window. Just as in the graph-based algorithm, the problem is the ordering relation for negation. Also, the ILP-based algorithm has to consider all possible combinations of event instances for the ordering relation. Assume the algorithm chooses the ordering relation $or(A, C)$ for negation. Then these four constraints need to be added to the ILP formulation:

$$C_1.time + x(C_1) < A_1.time + x(A_1)$$
$$C_1.time + x(C_1) < A_2.time + x(A_2)$$
$$C_2.time + x(C_2) < A_1.time + x(A_1)$$
$$C_2.time + x(C_2) < A_2.time + x(A_2)$$

4.4 Algorithm Comparison

Both presented algorithms have their advantages and disadvantages. The graph-based algorithm is complex but has more opportunities. In contrast, the ILP-based algorithm is simple but has limitations. A big advantage of the graph-based algorithm is the reordering condition. With the reordering condition the algorithm can decide which ordering relation of the private pattern is the best choice for the negation. Same holds for the avoidance of false-positives. Additionally, the graph-based algorithm has the opportunity to use different granularities for placing the event instances. But on the other hand, the generation of the graph is very complex, as well as choosing an ordering relation for negation. The ILP-based algorithm is much simpler. The generation of the ILP formulation is straight forward, but it is very restricted in consideration of the choice of the ordering relation for negation. The algorithm has to run the ILP to find out which ordering relation of the private pattern can be reordered in the defined ε-Range. Furthermore, the ILP-based algorithm cannot influence the granularity of arrival time changes of event instances like the graph-based algorithm. On the whole, the graph-based reordering algorithm seems to be the better choice considering the number of false-positives and false-negatives.

5 Evaluation

This chapter presents the results of the evaluation. The evaluation takes the graph-based reordering algorithm of Section 4.2, the ILP-based reordering algorithm of Section 4.3 and the event stream suppression approach [WHRN13] into account. Focus of the evaluation is on the number of false-positives and false-negatives as well as on the number of private patterns which are not concealed because of the restriction of the ε-Range. As already mentioned, this evaluation contains no experiments which examine the impact of the event stream reordering algorithms on the probability distribution according to the corresponding event type is generated. This is an interesting area for future work.

5.1 Setup and Parameters

The underlying CEP system is based on the framework of Ruben et al.[MKR15]. Each of the three approaches is implemented as an operator of the CEP system which gets the input event stream and returns the modified event stream as a result. The used linear programming solver is the *GNU Linear Programming Kit* (GLPK) for Java[15]. All experiments were performed on a machine with an Intel Core i5-4670 3.40GHz CPU and 16GB of RAM. The operating system is Windows 10 Pro with Java JRE 8.

Every experiment runs over a stream of 10.000 events. All events are generated according to a normal distribution with the inter arrival rate as mean and a standard deviation of 20%. The inter arrival rates of the event types are randomly chosen between 200ms and 1000ms. The event types participating in the queries and private patterns are also randomly chosen out of the set of available event types $\Sigma = \{A, B, C, \ldots\}$. For an easy comparison of the three algorithms, we developed a metric called *utility* which summarizes the quality of an algorithm referring to the number of introduced false-positives and false-negatives as well as to the number of private patterns which are not concealed in the modified event stream. The utility is defined as follows:

$$utility = \#matchingQueries - 2 * (\#falsePositives + \#matchingPrivatePatterns)$$

With this utility metric every introduced false-positive or false-negative leads to a utility penalty of -1. For every private pattern in the modified event stream, the utility is decreased by -2. Note that $\#matchingQueries$ already contains the penalty for the false-negatives because with every missing match the value $\#matchingQueries$ is not incremented. In contrast, every false-positive incremented $\#matchingQueries$. Therefore, we need to subtract two times the value of $\#falsePositives$ to achieve a utility penalty of -1. The optimal utility is the number of matching queries in the original event stream.

In the evaluation, the event stream suppression approach [WHRN13] is hard-constraint. Therefore, the algorithm will never report a private pattern match. Furthermore, this thesis does not consider event negation. That is why suppression never introduces any false-positives.

The evaluation examines the impact of the following parameters:

- ε-Range: The ε-Range is only a restriction for the reordering algorithms. It defines the range in which the corresponding event can be reordered.
- Size of the window: The size of the window impacts the number of event instances which has to be considered for a query or a private pattern match.
- Number of queries: The number of queries specifies the amount of public patterns which the algorithms need to take into account to prevent false-positives and false-negatives.
- Selectivity of queries: The selectivity of a query defines the size of a public pattern. For instance, query $Q = (A, B, C, D)$ has a selectivity of 4.
- Number of event types: The number of event type defines the size of the available set of different event types $\Sigma = \{A, B, C, \ldots\}$.

5.2 Evaluation of the ε-Range

One very interesting parameter for the number of concealed private patterns is the size of the ε-Range. Figure 5.1 shows the measured results of the evaluation referring to the ε-Range. The used parameters for the system are given by the following table:

Parameter	window size	#event types	#queries	#private patterns	selectivity
Value	500ms	5	3	1	3

The selectivity parameter specifies the maximum possible number of participating event types in a query. But every query has a selectivity of at least 2. The ε-Range varies between 5% and 75% of the inter arrival rate of the corresponding event. Note that different event types with different inter arrival rates will have different ε-Ranges.

The first chart in Figure 5.1 shows the utility in relation to the ε-Range. With increasing percentage for the ε-value, the utility of the reordering algorithms also increases. A low ε-value has the effect that reordering is not possible because of the big distance between the events. Therefore, the algorithms can only conceal a small number of private patterns. The second graph shows exactly this effect on the number of private patterns. With an increasing ε-Range, the number of reported private patterns decreases dramatically. With an ε-Range of at least 60%, the algorithms report no private patterns anymore. But this point also depends on the window size and is not a fixed value.

The thesis already discussed that the reordering of events can lead to false-positives and false-negatives. It is quite clear that the increasing number of concealed private patterns leads to the fact that also more false-positives and false-negatives are introduced. But the graphs in the second row show that the number of false-positives and false-negatives only increases very slow. Compared to the event suppression approach, the reordering algorithms achieve a higher utility with an ε-Range over at least 25% because the event suppression introduces such a huge number of false-negatives. In this evaluation, the suppression approach is hard-constraint and suppresses always one event of the matching private pattern. This declares the huge number of false-negatives. As already mentioned, the ε-Range has only an impact on the reordering algorithm. That is why the utility of the suppression approach stays constant.

The diagram in the third row shows that the ε-Range has no impact on the average execution time of a window.

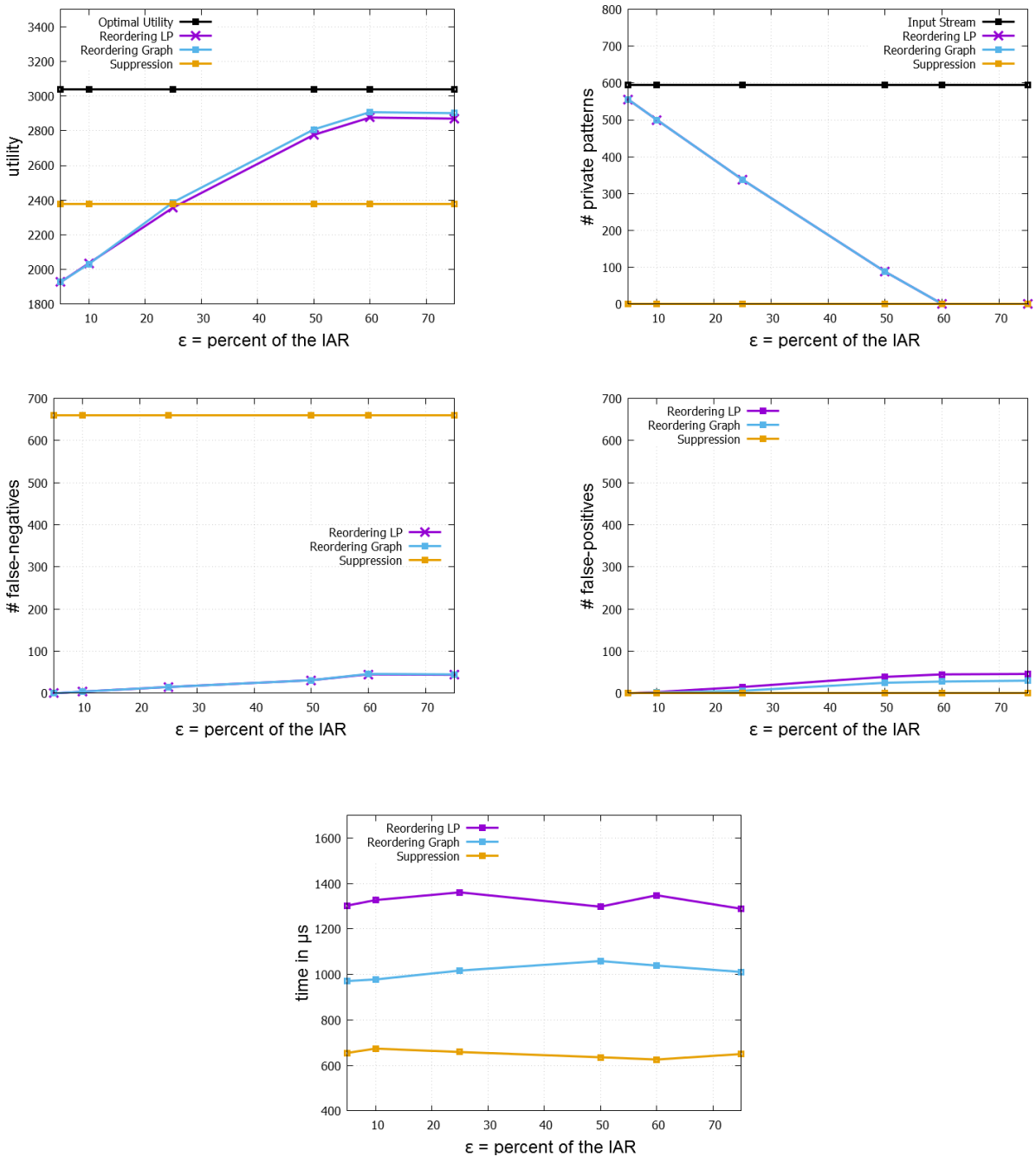


Figure 5.1: Evaluation of the ε-Range.

5.3 Evaluation of the Window Size

The size of the window has an impact on the number of event instances which need to be considered in one modification step. Figure 5.2 shows the results of the evaluation with the following constant parameters:

Parameter	#event types	#queries	#private patterns	selectivity	ε-Range
Value	5	3	1	3	40%

The first chart in Figure 5.2 shows the utility in relation to the window size. The utility of all algorithms increases up to a window size of $2000ms$. Afterwards, the utility of the reordering algorithms slowly decreases. The reason is shown in the second graph. The reordering algorithms report more private patterns in case of a bigger window. Reason is the ε-Range. An ε-Rang with 40% of the inter arrival rate which has a range of $200m$ to $1000ms$ is enough for a window size of $1000ms$ to $2000ms$. But afterwards, the ε-Rang limits the quality of the reordering algorithms in terms of concealed number of private patterns. The two graphs show that the reordering algorithms can achieve a higher utility as long as the ε-Rang is no big limitation. The graph-based reordering algorithm achieves a slightly higher utility than the ILP-based algorithm. The reason is that the graph-based algorithm avoids more false-positives.

The graphs in the second row show that both reordering algorithms introduce a small number of false-positives and false-negatives as long as the algorithms conceal a huge number of private patterns. If the reordering algorithms do not conceal private patterns, they also do not introduce false-positives or false-negatives. Equally as in the previous section, the event suppression approach leads to a huge number of false-negatives. With an increasing size of the window, the number of false-negatives also increases because more queries have a match in a window and the suppression of one event effects more matching queries.

Finally, the last chart shows that the window size has a huge impact on the average execution time for the reordering algorithms. It is quite clear that the increasing number of event instances in one window leads to a bigger graph and a bigger ILP formulation. The difference to the event suppression approach results from the fact discussed in Section 4.2.6 and Section 4.3.4. For the negation of an ordering relation the reordering algorithms have to take all instances of an event type into account.

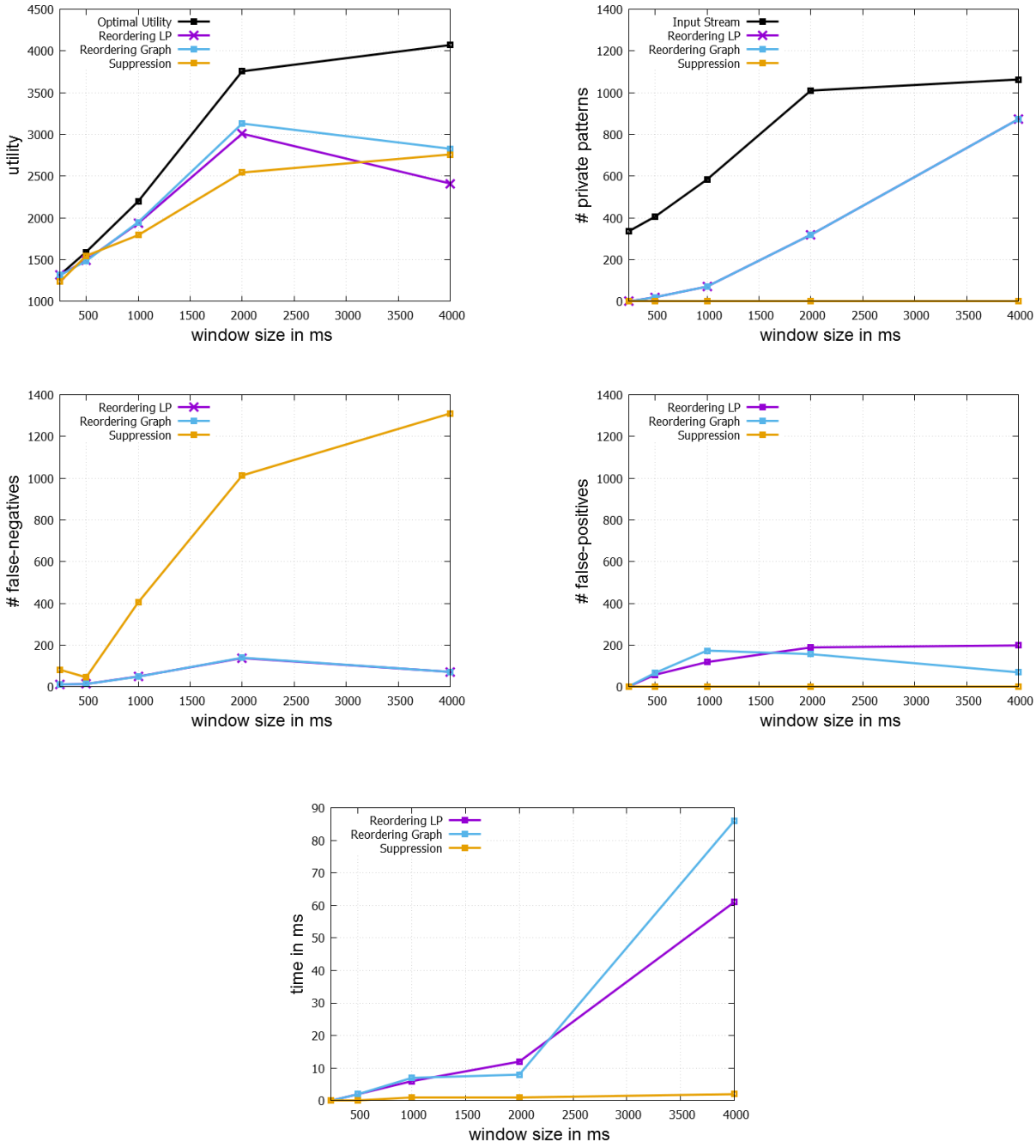


Figure 5.2: Evaluation of the window size.

5.4 Evaluation of the Number of Queries

Beside the concealing of private patterns, goal is also to prevent the loss of public patterns. With a higher number of such public patterns (queries), it is harder for the algorithms to conceal all private pattern matches without losing public patterns. The constant parameters have the following form:

Parameter	window size	#event types	#private patterns	selectivity	ε-Range
Value	500ms	15	1	3	40%

Figure 5.3 shows the results of this evaluation. This time, the utilities of all three algorithms are very close to each other. Until a total number of 16 queries, the utility is near by the optimal utility, but then it flattens. The two graphs in the second row show the reason. The number of false-positives and false-negatives increases after the total number of 16 queries stronger than before. Due to the big number of queries, the reordering algorithms cannot find a reordered event stream which prevents false-positives and false-negatives. Same holds for the suppression approach. With a higher number of queries it is more likely that all possible event types for suppression are also part of a query match. The second graph in the first row shows that the number of reported private patterns is not effected. The reordering algorithms report a small number of private patterns because of the ε-Range.

The last diagram in the third row shows that the number of queries has an substantial impact on the execution time of all algorithms. The impact on the suppression approach is very small. But the execution time of the reordering algorithms increases very strongly. The problem is that the graph grows for every matching query to avoid false-negatives and for every not matching query to avoid false-positives. Therefore, the number of matching queries is not critical because the graph-based algorithm considers the total number of queries anyway. Same holds for the ILP-based algorithm. Here the impact is not that strong because the ILP-based algorithm does not try to avoid false-positives explicitly.

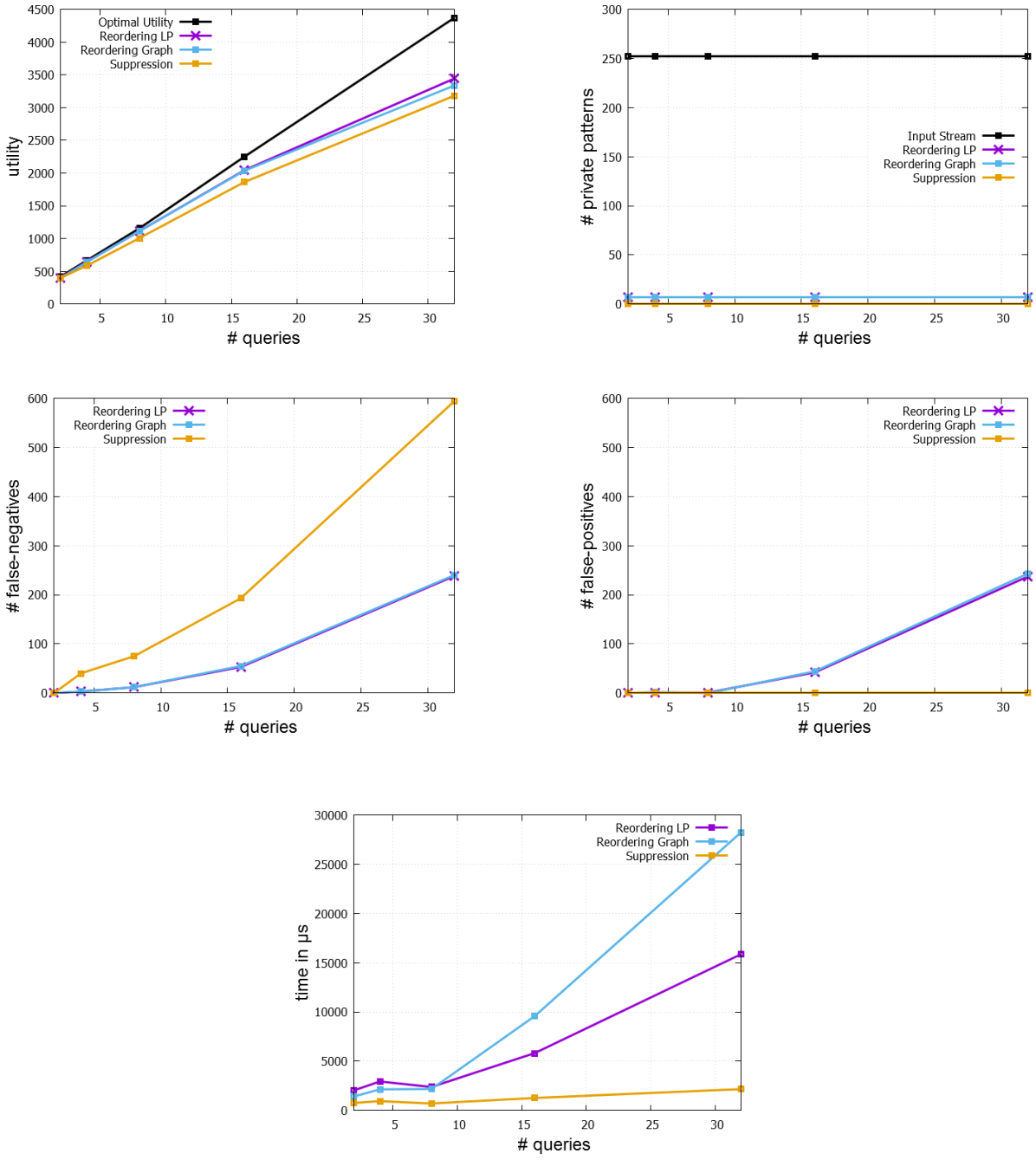


Figure 5.3: Evaluation of the number of queries.

5.5 Evaluation of the Selectivity of Patterns

The previous section showed the impact of the number of queries in the system. This section focuses on the selectivity of queries and private patterns. The selectivity defines the maximum possible number of event types which can participate in a public or private pattern. These are the used constant parameters:

Parameter	window size	#event types	#queries	#private patterns	ε-Range
Value	500ms	15	3	1	40%

The results of the evaluation are shown in Figure 5.4. Like in the previous sections, the first chart shows the utility of the algorithms compared to each other and the optimal utility. With a selectivity of already 5, all algorithms have only a utility near 0. But the optimal utility is also at a value near 0. Thus, the algorithms are quite good compared to the optimal solution. Reason for such a low utility value is that the queries and also the private pattern do not have a match on the event stream. With increasing selectivity, the corresponding patterns are getting more specific and matches get rare. The right chart shows the low number of private pattern matches. Therefore, the algorithms do not have to reorder or suppress an event which results in a low number of false-positives and false-negatives. This behavior is also shown in the diagrams of the second row. With a selectivity of 3, both reordering algorithms introduce a small number of false-positives and false-negatives but with a selectivity of already 5, the number decreases to a value near 0.

The same effect is shown by the last chart considering the average execution time for one window. The algorithms do not have to generate a graph or an ILP formulation because the input event stream has no match of a private pattern.

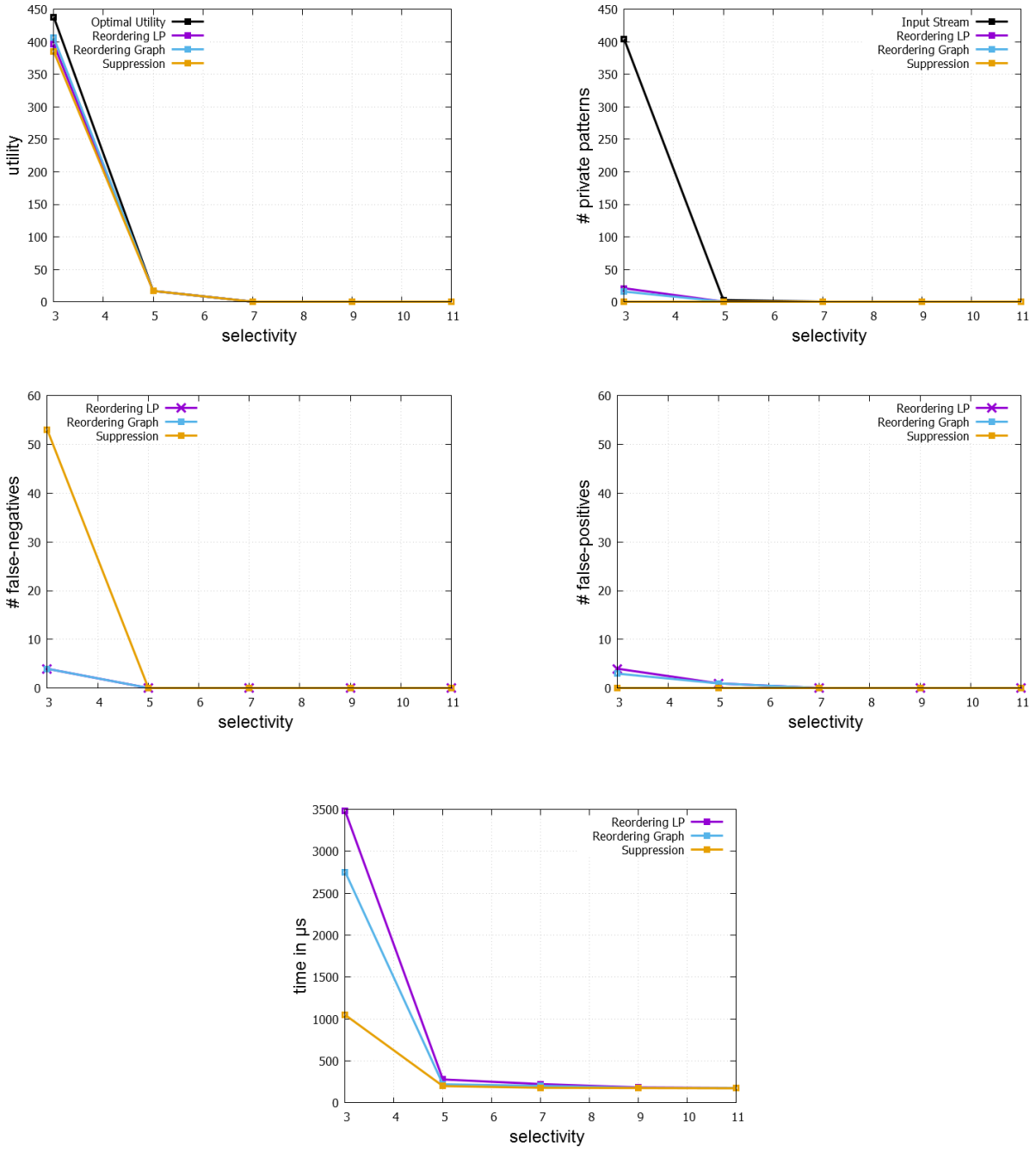


Figure 5.4: Evaluation of the selectivity of patterns.

5.6 Evaluation of the Number of Event Types

This section evaluates the impact on the size of the available set of different event types $\Sigma = \{A, B, C, \ldots\}$. The constant parameters are defined as follows:

Parameter	window size	#queries	#private patterns	selectivity	ε-Range
Value	500ms	3	1	3	40%

The evaluation of the number of different event types is shown in Figure 5.5. As always, the first chart shows the utility of the different algorithms. The utility of the reordering algorithm is very close to the optimal utility. Excluding the first part, also the utility of the suppression approach is quite close to the optimal. Reason for the difference in the first part can be seen in the diagram about the false-negatives and the private patterns. With a small number of different event types, the algorithms have to conceal a huge number of private patterns. To achieve that, the suppression approach introduces a high number of false-negatives which results in a lower utility. The reordering algorithms also conceal nearly all private patterns but only introduce a small number of false-positives and false-negatives.

The last chart in Figure 5.5 shows that the decreasing number of private patterns goes along with the decreasing average execution time because the reordering algorithms do not have to calculate a possible reordering and the suppression approach does not have to look for an event type for suppression.

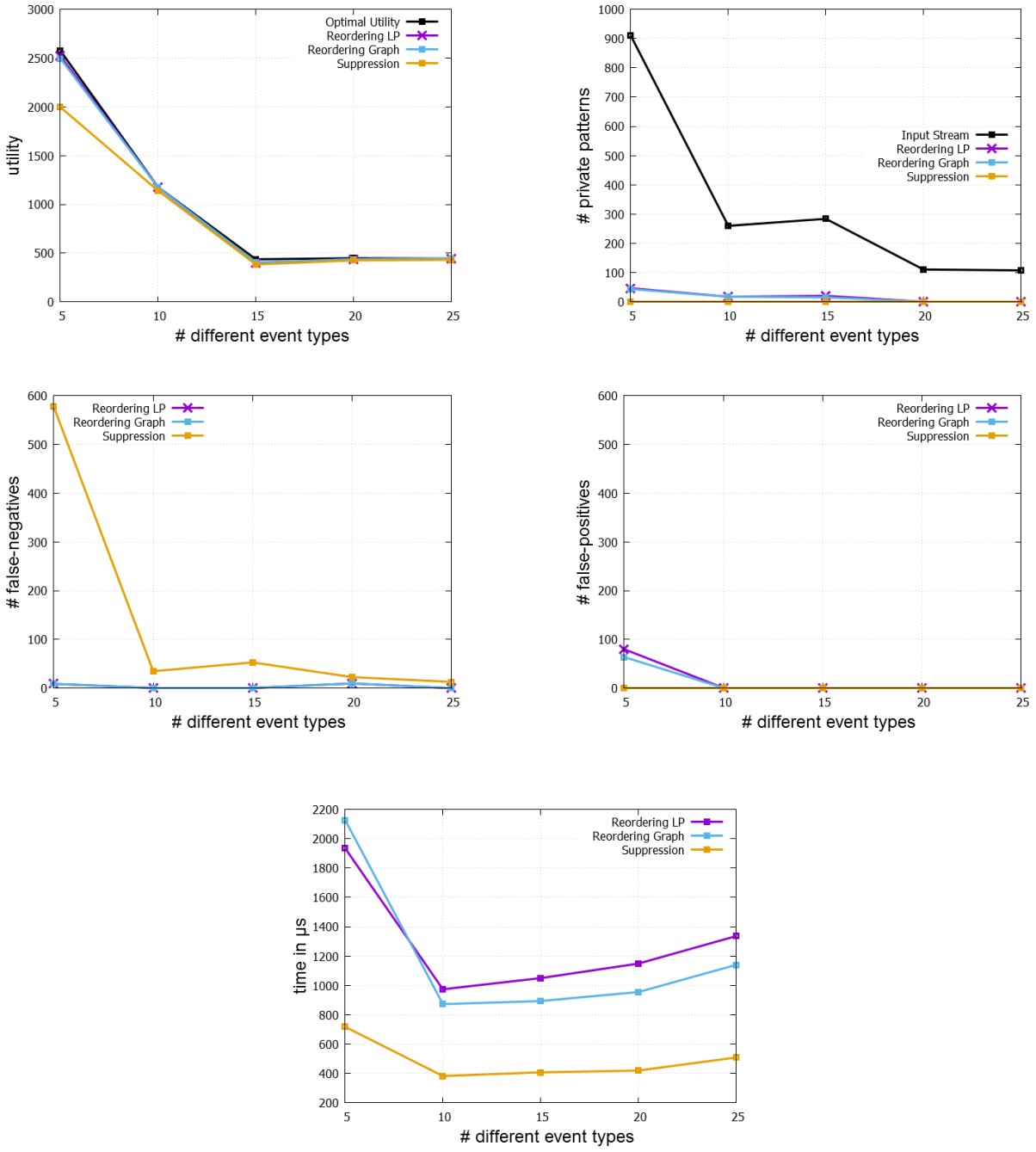


Figure 5.5: Evaluation of the number of event types.

5.7 Discussion

The evaluation shows that both reordering algorithms can conceal the most private patterns with a low number of introduced false-positives and false-negatives with a well defined ε-Range. As expected, the results of the graph-based algorithm and of the ILP-based algorithm are quite close to each other. They only differ in the number of introduced false-positives in which the graph-based algorithm introduces a slightly smaller number. Furthermore, the evaluation presents that in most cases the reordering algorithms achieve a higher utility than the event suppression approach. Problem is the introduced high number of false-negatives by the suppression approach. But benefit of the hard-constraint suppression approach is that it never reports a private pattern and it cannot introduce any false-positives as long as the patterns are not allowed to use negation. Furthermore, the execution time of the event suppression approach is much shorter as the one of the reordering algorithms.

Moreover, the evaluation presents that not all considered parameters have a direct impact on the algorithms. For instance, the evaluation of the selectivity of patterns shows rather its huge impact on the number of matches. This only effects the performance of the algorithms indirectly.

6 Conclusion and Future Work

This thesis considers the issue of *pattern-based access control* in event processing systems. Current access control mechanisms for event processing systems are not sufficient to prevent the revelation of private information in form of event patterns. Two algorithms focusing the problem of *pattern-based access control* are presented and evaluated in this thesis. Both algorithms try to reorder the input event stream in such a way that the information concerning critical privacy information is concealed and the public information is preserved. At the same time, the goal is to minimize the number of introduced false-positives and false-negatives.

Furthermore, this thesis examines the impact of the reordering algorithms on the inter arrival rate of events. For an attacker it might be possible to detect huge changes of the inter arrival rate in the modified event stream. With this additional knowledge, it could be possible to restore the original input event stream with the corresponding private patterns. Therefore, this thesis additionally presents the ε-Range to minimize the changes of the reordering approaches on the inter arrival rates of events. The ε-Range is a restriction to reorder an event only in its corresponding range. Due to this restriction, it is not always possible to find a reordered event stream which does not contain a private pattern.

Both developed reordering algorithms are evaluated and compared with an existing *pattern-based access control* approach which conceals private patterns by suppressing one of the participating events. The evaluation shows that the reordering algorithms can conceal the most private patterns with a low number of introduced false-positives and false-negatives despite of the restriction by the ε-Range. One big problem of the event suppression approach is that it introduces a very high number of false-positives. Therefore, the reordering algorithms in most cases achieve a higher utility than the event suppression approach.

Future Work

One interesting issue for the future work in the area of *pattern-based access control* in event processing systems is to examine the impact of event negation in public and private patterns. The suppression approach as well as both reordering algorithms do not consider event negation. Another issue is to examine the impact of the reordering algorithms on the probability distribution according to which an event type is generated. Closely related is the issue of finding a well-defined ε-Range.

Bibliography

[15] *GNU Linear Programming Kit for Java*. 2015. URL: http://glpk-java.sourceforge.net/ (cit. on p. 53).

[ABW06] A. Arasu, S. Babu, and J. Widom. "The CQL Continuous Query Language: Semantic Foundations and Query Execution." In: *The VLDB Journal* 15.2 (June 2006), pp. 121–142. URL: http://dx.doi.org/10.1007/s00778-004-0147-z (cit. on p. 7).

[ARX11] R. Adaikkalavan, I. Ray, and X. Xie. "Multilevel Secure Data Stream Processing." English. In: *Data and Applications Security and Privacy XXV*. Ed. by Y. Li. Vol. 6818. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 122–137. URL: http://dx.doi.org/10.1007/978-3-642-22348-8_11 (cit. on pp. 2, 8, 9).

[BBD+02] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. "Models and Issues in Data Stream Systems." In: *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS '02. Madison, Wisconsin: ACM, 2002, pp. 1–16. URL: http://doi.acm.org/10.1145/543613.543615 (cit. on p. 5).

[BTW+06] Y. Bai, H. Thakkar, H. Wang, C. Luo, and C. Zaniolo. "A Data Stream Language and System Designed for Power and Extensibility." In: *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*. CIKM '06. Arlington, Virginia, USA: ACM, 2006, pp. 337–346. URL: http://doi.acm.org/10.1145/1183614.1183664 (cit. on p. 7).

[BW01] S. Babu and J. Widom. "Continuous Queries over Data Streams." In: *SIGMOD Rec.* 30.3 (Sept. 2001), pp. 109–120. URL: http://doi.acm.org/10.1145/603867.603884 (cit. on p. 6).

[CCFT09] J. Cao, B. Carminati, E. Ferrari, and K.-L. Tan. "ACStream: Enforcing Access Control over Data Streams." In: *Data Engineering, 2009. ICDE '09. IEEE 25th International Conference on*. Mar. 2009, pp. 1495–1498 (cit. on pp. 2, 8, 9).

[CM10] G. Cugola and A. Margara. “TESLA: A Formally Defined Event Specification Language.” In: *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems*. DEBS ’10. Cambridge, United Kingdom: ACM, 2010, pp. 50–61. URL: http://doi.acm.org/10.1145/1827418.1827427 (cit. on pp. 6–8).

[CM12a] G. Cugola and A. Margara. “Processing Flows of Information: From Data Stream to Complex Event Processing.” In: *ACM Comput. Surv.* 44.3 (June 2012), 15:1–15:62. URL: http://doi.acm.org/10.1145/2187671.2187677 (cit. on p. 1).

[CM12b] G. Cugola and A. Margara. “Processing Flows of Information: From Data Stream to Complex Event Processing.” In: *ACM Comput. Surv.* 44.3 (June 2012), 15:1–15:62. URL: http://doi.acm.org/10.1145/2187671.2187677 (cit. on pp. 5, 6, 14, 16, 18, 19).

[CM13] G. Cugola and A. Margara. “Deployment strategies for distributed complex event processing.” English. In: *Computing* 95.2 (2013), pp. 129–156. URL: http://dx.doi.org/10.1007/s00607-012-0217-9 (cit. on pp. 1, 14, 17).

[GÖ03] L. Golab and M. T. Özsu. “Issues in Data Stream Management.” In: *SIGMOD Rec.* 32.2 (June 2003), pp. 5–14. URL: http://doi.acm.org/10.1145/776985.776986 (cit. on p. 14).

[Int14] International Data Corporation (IDC). *The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things*. Apr. 2014. URL: http://www.emc.com/leadership/digital-universe/2014iview/internet-of-things.htm (cit. on p. 1).

[Kah62] A. B. Kahn. “Topological Sorting of Large Networks.” In: *Commun. ACM* 5.11 (Nov. 1962), pp. 558–562. URL: http://doi.acm.org/10.1145/368996.369025 (cit. on pp. vii, 40).

[Luc01] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001 (cit. on p. 5).

[MKR15] R. Mayer, B. Koldehofe, and K. Rothermel. “Predictable Low-Latency Event Detection With Parallel Complex Event Processing.” In: *Internet of Things Journal, IEEE* 2.4 (Aug. 2015), pp. 274–286 (cit. on pp. 8, 53).

[MPE+10] M. Migliavacca, I. Papagiannis, D. M. Eyers, B. Shand, J. Bacon, and P. Pietzuch. “DEFCon: High-Performance Event Processing with Information Security.” In: *USENIX Annual Technical Conference (USENIX ATC’10)*. USENIX. Boston, MA, USA: USENIX, June 2010. URL: http://www.doc.ic.ac.uk/~prp/doc/research/sf-usenix10-camera.pdf (cit. on pp. 2, 8, 9).

[Raq12] Raquel. SIMUL8 blog. 2012. URL: http://blog.simul8.com/simul8-tip-whats-the-difference-between-arrival-rates-and-inter-arrival-times/ (cit. on pp. 2, 31).

[SKRR13] B. Schilling, B. Koldehofe, K. Rothermel, and U. Ramachandran. "Access Policy Consolidation for Event Processing Systems." In: *Networked Systems (NetSys), 2013 Conference on*. Mar. 2013, pp. 92–101 (cit. on pp. 2, 8, 9).

[Weia] E. W. Weisstein. *Mean*. From MathWorld. A Wolfram Web Resource. URL: http://mathworld.wolfram.com/Mean.html (cit. on pp. 34, 36).

[Weib] E. W. Weisstein. *Normal Distribution*. From MathWorld. A Wolfram Web Resource. URL: http://mathworld.wolfram.com/NormalDistribution.html (cit. on p. 33).

[Weic] E. W. Weisstein. *Standard Deviation*. From MathWorld. A Wolfram Web Resource. URL: http://mathworld.wolfram.com/StandardDeviation.html (cit. on pp. 34, 36).

[Weid] E. W. Weisstein. *Variance*. From MathWorld. A Wolfram Web Resource. URL: http://mathworld.wolfram.com/Variance.html (cit. on pp. 21, 36).

[WHRN13] D. Wang, Y. He, E. Rundensteiner, and J. F. Naughton. "Utility-maximizing Event Stream Suppression." In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. SIGMOD '13. New York, New York, USA: ACM, 2013, pp. 589–600. URL: http://doi.acm.org/10.1145/2463676.2465305 (cit. on pp. 2–5, 8–11, 13, 15, 16, 20, 21, 23, 27, 30, 53, 54).

[XRAG13] X. Xie, I. Ray, R. Adaikkalavan, and R. Gamble. "Information Flow Control for Stream Processing in Clouds." In: *Proceedings of the 18th ACM Symposium on Access Control Models and Technologies*. SACMAT '13. Amsterdam, The Netherlands: ACM, 2013, pp. 89–100. URL: http://doi.acm.org/10.1145/2462410.2463205 (cit. on pp. 2, 8, 9).

All links were last followed on January 13, 2016.

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

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift