1 Dynamic Pipeline

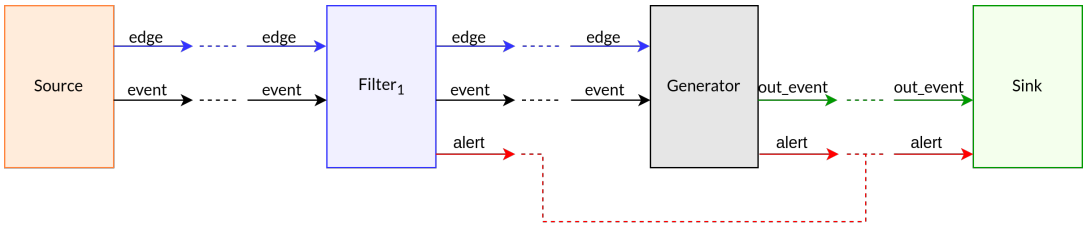

Figure 1: Pipeline Schema

Description of the channels:

- `edge`: only edge dedicated channel
- `event`: events channel
- `alert`: direct channel from the filters (in particular the filter worker) to the sink (it does not go through the Generator, although it has it to be able to give it to the filters so that they are able to write on it)
- `out_event`: direct dedicated event channel between Generator and Sink.

2 Fraud Patterns

2.1 Fraud Pattern I - Card Cloning

A first algorithmic proposal to detect this kind of fraud pattern is the one shown in the algorithm **??**. Note that S refers to the filter's subgraph and e_{new} is the new incoming edge belonging to the filter, such that it is a opening interaction edge, since in the case it is a closing interaction edge, we do not perform the CheckFraud().

Algorithm 1 CheckFraud(S, e_{new}) – **initial version**

Require: S is the subgraph of edges of the filter (sorted by time)
Require: e_{new} is the new incoming opening interaction edge belonging to the filter

```
1: fraudIndicator ← False
2: i ← |S|
3: while i > 0 and fraudIndicator = False do
4:    e_i ← S[i]
5:    t_min ← obtain_t_min(e_i, e_new)
6:    t_diff ← e_new.start − e_i.end
7:    if t_diff < t_min then
8:       createAlert(e_i, e_new)
9:       fraudIndicator ← True
10:   end if
11:   i ← i − 1
12: end while
```

There are some aspects and decisions of this algorithm that are worth to describe:

- **Pairwise detection**. The checking of the anomalous fraud scenario is done doing the check between the new incoming edge e_{new} and each of the edges e_i of the filter's subgraph S.
- **Backwards order checking**. The pairs (e_{new}, e_i) are checked in a backwards traversal order of the edge list of the subgraph S, starting with the most recent edge of the subgraph and ending with the oldest.
- **Stop the checking whenever the first anomalous scenario is detected**. Whenever an anomalous scenario corresponding to a pair (e_{new}, e_i), then we stop the checking at this point and emit the corresponding alert. Therefore we do not continue the checking with previous edges of S.
- **Emission of the pair** (e_{new}, e_i) **as the alert**. The alert is composed by the pair (e_{new}, e_i) that is detected to cause the anomalous scenario. Both edges are emitted in the alert since we do not know which is the one that is the anomalous. On the one hand, it can be e_i, which is previous to e_{new}, in the case that e_i at the moment it arrived it did not cause any alert with the previous edges/transactions of the subgraph and it causes it now with a new incoming edge e_{new} which is a regular transaction of the client. On the other hand, it can be e_{new}, which is the last having arrived to the system, that it directly causes the alert with the last (ordinary) transaction of the card.

However, a more detailed study, lead us to a simplification of the initially proposed algorithm to the one shown in **??**. On it we just perform the checking between the new incoming edge e_{new} and the most recent edge of the subgraph S, e_{last}.

Algorithm 2 CheckFraud(S, e_{new}) – **definitive version**

Require: S is the subgraph of edges of the filter (sorted by time)
Require: e_{new} is the new incoming opening interaction edge belonging to the filter

```
1: last ← |S|
2: e_last ← S[last]
3: t_min ← obtain_t_min(e_last, e_new)
4: t_diff ← e_new.start − e_last.end
5: if t_diff < t_min then
6:     createAlert(e_last, e_new)
7: end if
```

In what follows we argument the reason why it is sufficient to just check the fraud scenario among e_{new} and the last/most recent edge of the subgraph and not have to continue having to traverse the full list of edges.

Assume that we have a subgraph as depicted in Figure **??**, and that we do not know if there have been anomalous scenarios produced between previous pairs of edges of the subgraph. Name $F_I(y_i, y_j)$ a boolean function that is able to say whether it exists an anomalous fraud scenario of this type between the pair of edges (y_i, y_j) or not. In addition, note that the edges of the subgraph S are ordered by time in ascending order, in such a way that $y_1 < y_2 < y_3$. Finally note that $y_3 \equiv e_{new}$ as it is the new incoming edge and $y_2 \equiv e_{last}$, since it is the last edge / the most recent edge of S.

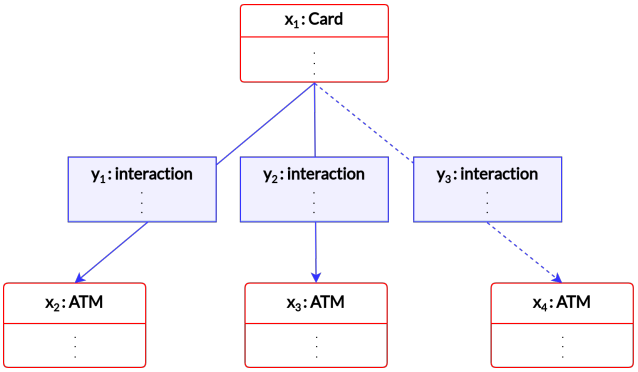

Figure 2: Subgraph S of a card – Fraud Pattern I

Note that we can have that:

- $F_I(y_2, y_3)$: We emit an alert of this anomalous scenario produced between the pair (y_2, y_3). We could continue checking for anomalous scenarios between y_3 and previous edges of the subgraph. However, what we consider important for the bank system is to detect the occurrence of an anomalous scenario in a certain card. Therefore, we consider that, to achieve this, it is enough to emit a single alert of anomalous scenario on this card, and not many related with the same incoming transaction of the same card.

- $\neg F_I(y_2, y_3)$: We analyze whether it would be interesting or not to continue the checking with previous edges of the subgraph, based on assumptions on the fraud checking between previous edges. In particular we can have two cases:
  - If $F_I(y_1, y_2)$: Having this it can happen that either $F_I(y_1, y_3)$ or $\neg F_I(y_1, y_3)$. In the case of $F_I(y_1, y_3)$, since $\neg F_I(y_2, y_3)$, we can infer that the anomalous scenario detected between y_1 and y_3 is a continuation of the same previous anomalous scenario detected between y_1 and y_2. Therefore, we can conclude that this does not constitute a new anomalous scenario that would require an alert.
  - If $\neg F_I(y_1, y_2)$: It can be shown that *by transitivity*, having $\neg F_I(y_1, y_2) \wedge \neg F_I(y_2, y_3) \implies \neg F_I(y_1, y_3)$.
    TODO: Show a formal demostration of this case!

Therefore, we have seen that, it is enough to perform the checking between the pair formed by e_{new} and the most recent edge of the subgraph e_{last}. □

TODO: Complete other aspects of the filter worker algorithmic workflow
Others – not so much related with the CheckFraud algorithm, but in general with the filter's algorithm –:

- Save all the edges in the subgraph S, even though they are the reason of the creation of an anomalous scenario.
- Number of anomalous fraud scenarios that can be detected. Bounded by:

$$\#TX\_ANOM \leq SCENARIOS \leq 2 * \#TX\_ANOM$$