1 Dynamic Pipeline

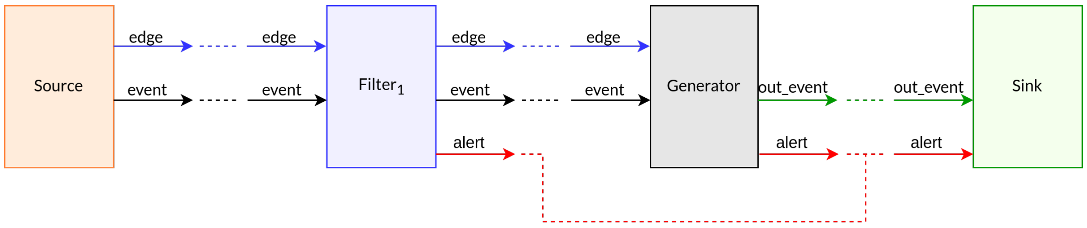

Figure 1: Pipeline Schema

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

So far, the algorithm (pseudocode) 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})

```
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: $i \leftarrow |S|$
 3: while $i > 0$ and fraudIndicator = False do
 4:     $e_i \leftarrow S[i]$
 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:     $i \leftarrow i - 1$
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. In what follows we argument the reason why it is sufficient to stop the checking at this point, not having to traverse the full list of edges. Assume that we have a subgraph as depicted in Figure x, and that we do not know if there have been anomalous scenarios produced between previous pairs of edges of the subgraph. Name $F_I(X, Y)$ a boolean function that is able to say whether it exists an anomalous fraud scenario of this type between the pair of edges (X, Y) or not.

  We can have that:

  - $F_I(B, C)$: ...
  - $\neg F_I(B, C)$: ..., 2 possible cases:
    - $F_I(A, B)$
    - $\neg F_I(A, B)$
- **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.

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.